Hierarchical Simulation of Timed Behaviours of Structured Occurrence Nets

Salma Alharbi

Supervisor: Prof. Maciej Koutny

School of Computing

Newcastle University

A thesis submitted for the degree of

Doctor of Philosophy

September 2024

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Salma Alharbi

September 2024

Acknowledgements

To begin, I want to thank Allah the Almighty for all the good things that have happened to me throughout this journey.

I would like to express my sincere gratitude to my supervisor, Prof. Maciej Koutny, for all of his help and support during the writing of this thesis. Their tremendous support and insightful advice have greatly influenced the direction of my work. His encouragement and invaluable insights were crucial in shaping this work. Your input and comments throughout the completion of this thesis are tremendously appreciated. Thank you.

I would like to thank our SON group memebrs. I would especially want to thank Prof. Brian Randell for his insightful conversations. Also acknowledged with gratitude is to Dr. Anirban Bhattacharyya for his invaluable assistance and insightful remarks. Also a special thanks to Dr. Nadiyah Almutairi for her support, feedback and discussion of my Thesis. Also, I would like to thank Mohammed Alahmadi and Tuwailaa Alshammari for their comments and feedback throughout my research.

I would like to express my deepest gratitude to my husband, [Ahmed], I am so thankful for all the time you've spent being patient, supportive, and understanding. Your believing in me has been my biggest source of strength. It provided the motivation and strength needed to get through the difficulties and challenges along my journy. I would want to express my sincere appreciation to my beloved infant, [Battal]. My will to succeed and achieve has been motivated by your existence. During my journy of this thesis, I have found much-needed breaks and refreshment from your smiles, embraces, and lighthearted moments.

My sincere appreciation goes out to my parents for their everlasting love, support, and encouragement during the process of finishing this thesis. I can not forget to thank my brothers and sister for their love and support. Thank you.

Abstract

A complex evolving system (CE-system) is composed of a large number of (sub-)systems concurrently interacting among themselves and with the system's environment. A CE-system is subject to modifications by other systems. Typical examples of CE-systems are large distributed systems whose software is continually updated, or dynamically evolving (cyber)crime investigations. Time simulation is a powerful tool used in many fields to model and analyse the behaviour of CE-systems, such as crimes and accidents. In crime investigations, time simulation may be an effective technique since it enables investigators to piece together the sequence of events that led to a crime. Investigators may better grasp the circumstances leading up to a crime by simulating crucial events happening in time, which can be useful in identifying suspects and acquiring evidence.

A typical notation for recording the behaviour of a CE-system is some form of a directed acyclic graph. The framework of structured occurrence nets (SO-nets) can play an important role in the representation of CE-system behaviours. In general, SO-nets are sets of related acyclic Petri nets, employing different types of formally defined relationships and supporting various types of hierarchy and abstraction, which represent the details of concurrency and causality relations between executed events.

This thesis focuses on two objectives. First, it seeks to develop theoretical underpinnings, new algorithms, and implemented prototype software tools for hierarchical and abstraction-based analyses and simulations of timed behaviours within CE-systems represented using SO-nets. Second, it aims to analyse scenarios in which different time granularities are utilized at different levels of abstraction. This involves developing theoretical frameworks, algorithms, and prototype software tools tailored to aforementioned specific requirements. The discussion

is carried out using behavioural structured acyclic nets (BSA-nets) that are part of the SO-nets framework. The overall research question to be addressed is whether the resulting framework can provide effective support for incident investigators.

Contents

List of Figures

Chapter 1

Introduction

This chapter provides an introduction to the thesis, outlining its aims, objectives, and overall structure. Additionally, it includes an outline of the thesis and a list of relevant publications.

1.1 Motivation

A complex evolving system (CE-system) is composed of a large number of concurrently-acting (sub)systems, interacting with each other and with the system's environment, and that is subject to modification by other systems. Typical examples are large distributed systems whose software is continually updated, and dynamically evolving crime investigations. Such very diverse 'event-based' systems all suffer from a very high complexity of both design and behaviour due to the need to cope with a vast number of recordable events or facts and the resulting explosion of combinatorial state space, complex relationships between the state information representation and the dynamic evolution and reconfiguration of the system. Structure plays a crucial role in assisting designers in managing design complexity, particularly in the fields of software engineering and hardware design domains [47]. The effective use of structuring notations greatly reduces the cognitive complexity of designs and the resources involved in their representation and manipulation. The purpose of a system design is to define how a system will behave. Notations for recording actual or potential system behaviour have not been attracted levels of interest comparable to those for system

design notations [16]. This is probably because detailed records of the behaviour of complex systems are mainly used out of sight within tools for system visualisation, verification, synthesis, and failure analysis rather than in documents and user interfaces. Three key areas that need to be understood and supported to manage the cognitive complexity of CE-system behaviours are: (i) the choice of an appropriate notation, (ii) the range of tasks and activities involved in the system design process in which this notation would have an instrumental role, and (iii) the appropriate tool support [47].

The typical notation for recording the behaviour of an asynchronous system is some form of directed acyclic graph. The most developed of such notations is that of an occurrence net (O-net). An occurrence net is an acyclic net that provides a complete and unambiguous record of all the causal dependencies among the events involved, including both backward (more than one arrow incoming to a place) and forward non-determinism (more than one arrow outgoing from a place). It represents a singular 'causal history' and has a single final marking [20]. The formalism of structured occurrence nets (SO-nets) can play a similarly important role regarding the representation of CE-system behaviours. SO-nets are sets of related O-nets that capture causality and concurrency information about a single execution of a system, employing different types of formally defined relations and supporting various types of abstraction, including (behavioural structured occurrence net (BSO-nets), communication structured occurrence nets (CSO-nets)) and alternative occurrence net (ALTO-nets) [49, 46]. Further, previous research on SO-nets has created a system for timed behaviours [23, 5]. In [11], a SAT-based model checking for communication structured acyclic nets was proposed, and adding probabilistic analysis to communication structured acyclic nets was first discussed in [10].

Time is an important and basic entity that affects every part of our lives. It tells us how long something lasts, how far something goes, and how much it changes. It affects almost every part of our lives. Time forms our experiences, how we see the world, and how we interact with it. It does this through the cycles of day and night and the constant forward motion of seconds, minutes, and hours. Therefore, representing (date-time) information about such systems is important. Timed SO-nets are based on groups of associated timed

occurrence nets and are designed for reasoning about related events and causality with time information that is uncertain or missing in evolving systems. When modelling a system based on time, such as accidents or crimes, it is important to identify the order in which events have happened and to identify the duration of the events. Time is a crucial measure for studying and understanding how complex systems behave. It allows for the analysis of changes that occur over time and discover the patterns that exist under the surface.

Therefore, the development of time simulation has become essential in contemporary study, providing a powerful method for understanding complex systems and predicting their behaviour under different conditions. By simulating the flow of time, researchers and practitioners can gain insights into how systems might behave in the future and develop strategies for optimising their performance. Time simulation is used in many fields to model and analyse the behaviour of complex systems over time, such as crimes and accidents. In criminal investigations, time simulation may be an effective technique, as it enables investigators to piece together the sequence of events that led to a crime. Investigators may better grasp the circumstances leading up to a crime by simulating time, which can be useful in acquiring evidence and identifying suspects [16]. In a real-time simulation, the simulation is run in discrete time with constant steps, also known as fixed step simulation, as time moves forward in equal duration of time. Discrete time simulation is used in time Petri net (Petri nets are a mathematical model used to describe and analyse the behaviour of concurrent systems). In [13] the simulations of dynamic discrete event systems are represented using a timed transition Petri net model. Also time simulation for Petri nets were introduced in [68, 52, 19, 26, 40].

In this thesis, we develop the theoretical underpinning and tool support for time and time simulation based on BSO-nets, CSO-nets, and ALTO-net. These abstractions are derived from the underlying framework of SO-nets, which were introduced in [49].

In addition, analyses and simulations of timed behaviours of CE-systems in the case when different time granularity is used at different levels of abstraction. Time granularity is the degree of accuracy or precision with which we measure time. In this thesis, we extend the work to include the time granularity in behavioural structured acyclic net (BSA-nets), which

is a part of structured occurrence net framework. The level of accuracy or precision with which we calculate the passage of time is referred to as time granularity. To conduct efficient assessments of complex systems, it is especially important to have the ability to divide time into smaller chunks. For instance, in studying the behaviour of a system, measuring time in seconds may be the most acceptable method for analysing low-level (detailed) system behaviour, but measuring time in minutes may be the most appropriate method for analysing high-level (abstracted) system behaviour[80]. Nevertheless, it is of the utmost significance that the information regarding time that is gathered through the utilisation of various granularity time scales be maintained consistently throughout the various levels of abstraction.

Behavioural structured acyclic nets (BSA-nets) model activities of evolving systems [4]. An execution history is represented on two different levels: the lower-level, which is used to indicate behavioural details, and the upper-level, which is used to represent the stages (phases) of system evolution. Figure 1.1 shows an example of BSA-net [4]. To keep the technical discussion in this thesis simpler, we do not consider the general BSA-nets. Instead, we consider the case in which the upper-level comprises one acyclic net, and the lower-level comprises one acyclic net.

This thesis will extend the SO-nets model theory and its tool support to provide novel algorithms and implementations to allow the simulation of timed behaviours of CE-system in meaningful ways. The work will deliver an extension of WORKCRAFT plug-in, which is a tool that provides a flexible common underpinning for graph based models, and its plug-in SONCRAFT provides some initial facilities for entering, editing, validating, and simulating SO-nets. We also extend the concept of behavioural abstraction of BSA-nets to support a various time unit of multiple levels.

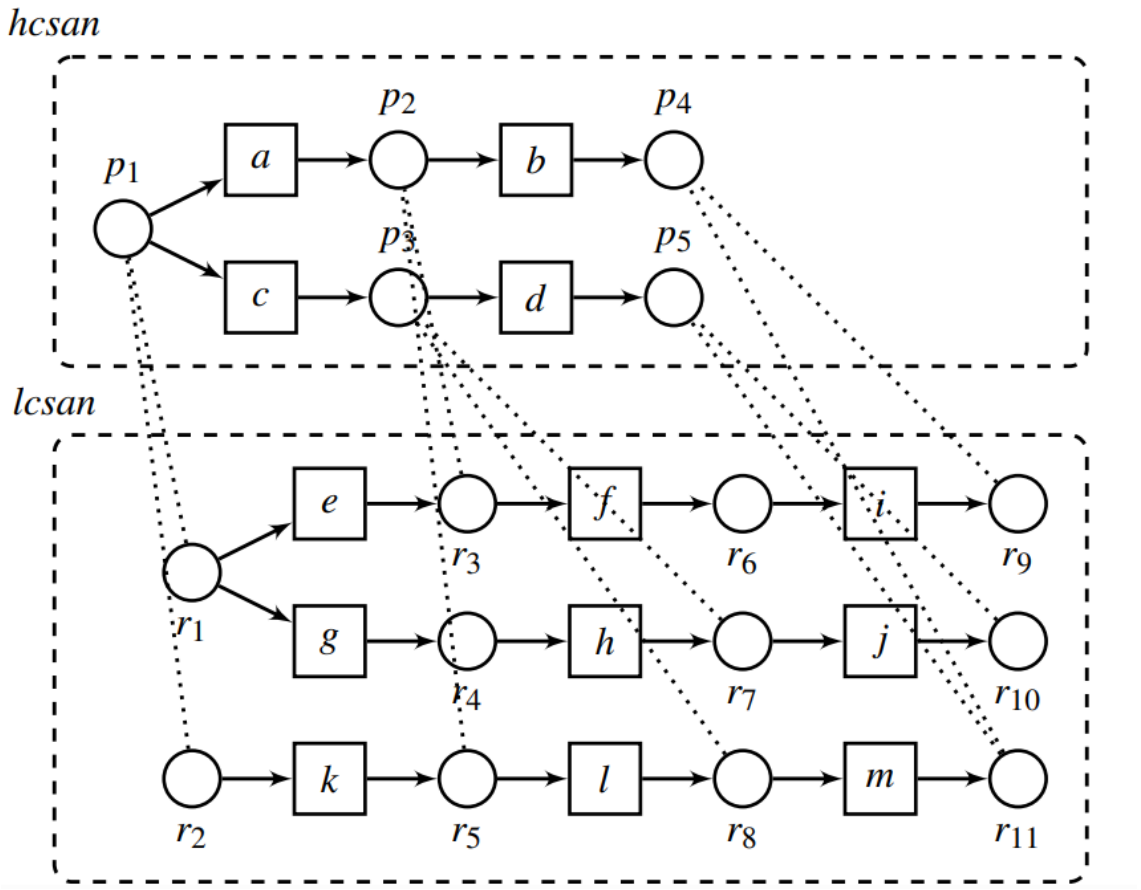


Figure 1.1 BSA-net with one lower-level acyclic net and one upper-level acyclic net.

1.2 Aims and objectives

1.2.1 Research aim

This thesis aims to present theoretical underpinnings, new algorithms, and prototype software implementation for hierarchical and abstraction-based analyses and simulations of timed behaviours of complex evolving systems using SO-nets. Additionally, it aims to develop theoretical underpinnings, new algorithms, and prototype software tools for similar analyses and simulations, considering scenarios in which different time granularities are utilized across various levels of abstraction, using the behavioural structured acyclic nets, which is a part of the framework of structured occurrence nets.

1.2.2 Research objectives

Developing the time property information for SO-nets. We extend the existing basic SO-net model and ALTO-net to include (date-time) intervals. We present a formal description of how to verify the consistency of timing information provided in a SO-net and how to determine the time information estimation of a specific node or the entire SO-net by utilising causal relations. Time-based algorithms for time estimation and consistency checking are introduced.

Extending the concept of (date-time) property information. We introduce a time simulation algorithm to simulate the time behaviour of the model using SO-nets. New execution time semantics are introduced for variants of SO-net in order to a step-by-step time simulation. We also provide a new algorithm for maximal firing steps with time semantics for SO-nets. The evaluation of the proposed solution and validation of its effectiveness will be carried out using carefully designed small size example, the existing medium size case study constructed by an MSc student [38].

Designing and implementing new tool in SONCRAFT plug-ins. The prototype tools will be implemented as SONCRAFT plug-ins, implementing the ideas and algorithms developed in the formal methods part of Chapters 3 and 4. The prototype tool implementation and evaluation will combine and implement the time property for SO-nets and ALTO-net, specific time simulation algorithms developed in theoretical investigation with suitably adapted selected simulation algorithms found in the commercial software tools. Further, algorithms checking the consistency and correctness of the new abstraction methods will be incorporated in the prototype tool.

Extending the concept of behavioural abstraction to support time granularity. We investigate the time granularity to develop theoretical underpinnings and new algorithms for hierarchical and abstraction-based analyses and simulations of timed behaviours for cases in which different time granularity is used at different levels of abstraction using BSA-nets that are part of the SO-nets framework.

Designing and implementing a tool for the concept of time granularity. We provide the implementation of the prototype software tool for BSA-nets by adding different time granularity (*minute* and *second*) and simulations of timed behaviours using the algorithms developed in the theoretical investigation.

1.3 Outline of the thesis

The structure of the thesis is as follows:

Chapter 2 presents the background about SO-net and their abstractions (BSO-net, CSO-net and ALTO-net). The chapter also gives an overview of the time simulation and its techniques found in the commercial software tools that are useful to simulate the time behaviours. A background on time granularity is also provided.

Chapter 3 describes SO-nets with time property notations and algorithms for time estimation and for checking consistency for all SO-nets abstractions and ALTO-net. The chapter also outlines the related works of time property in Petri net.

Chapter 4 describes time simulation for SO-nets and their abstractions BSO-net, CSO-net, and ALTO-net. Further, the chapter gives the time semantics theory and algorithm and the design of a small case study. It also outlines the related works on time simulation.

Chapter 5 introduces the SONCRAFT plug-ins framework and implementation by providing additional tools for time-based functionality and time simulation.

Chapter 6 presents the background about BSA-nets. Additionally the chapter presents the time granularity in BSA-nets model with multiple time unit (*minute* and *second*) representations. The chapter also discusses the time granularity theory, algorithm and time simulation steps using different levels of time unit. It provides a prototype tool time granularity and time granularity simulation for BSA-nets. It outlines the related works on time granularity.

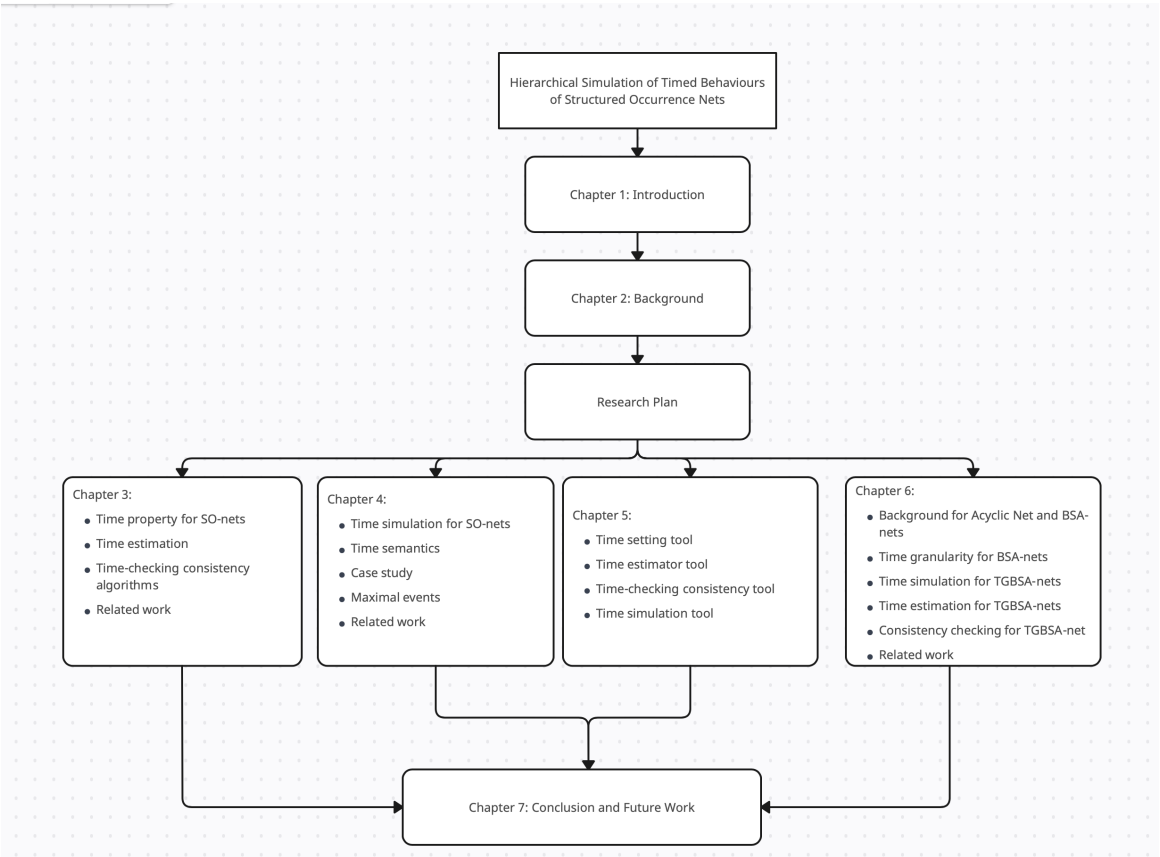


Figure 1.2 Overall conceptual structure of the thesis

Chapter 7 concludes and summarises the work and provides recommendations for future studies.

This thesis addresses the study subjects outlined above. The research methodology that will be employed to achieve the aims and objectives of this thesis is based on theoretical investigation and prototype tool implementation and evaluation. Figure 1.2 is a graphical depiction of the fundamental conceptual framework of this thesis.

1.4 List of publications

Publications listed below provide documentation of this thesis's objectives:

1. Alharbi, S. (2023). Hierarchical simulation of timed behaviours of structured occurrence nets. In Proceedings of the 2023 International Workshop on Petri Nets and

Software Engineering (PNSE 2023) co-located with the 44th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2023)

2. Alahmadi, M., Alharbi, S., Alharbi, T., Almutairi, N., Alshammari, T., Bhattacharyya, A., Koutny, M., Li, B. and Randell, B. (2024) Structured Acyclic Nets. arXiv preprint arXiv:2401.07308

3. Alharbi, S. (2024). Time granularity in behavioural structured acyclic nets. In 2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC), pages 1–12. IEEE.

Chapter 2

Background

This chapter outlines the background of structured occurrence nets (in short SO-net) and all their abstractions CSO-nets, BSO-nets and ALTO-net. In addition, we provide an overview of time simulations and their techniques. Finally, we outline the background of time granularity.

2.1 Introduction

A complex evolving system (CE-system) is composed of a large number of concurrently-acting (sub)systems interacting with each other and with the system environment, and that is subject to modification by other systems. Typical examples are large distributed systems whose software is continually updated, or dynamically evolving crime investigations. Such diverse event-based systems all suffer from a very high complexity of both design and behaviour due to the need to cope with a vast number of recordable events or facts and the resulting explosion of combinatorial state space, complex relationships between the state information representation and the dynamic evolution and reconfiguration of the system.

Structure plays a crucial role in assisting designers in managing design complexity, particularly in the fields of software engineering and hardware design domains. The effective use of structuring notations greatly reduces the cognitive complexity of designs, and the resources involved in their representation and manipulation. The purpose of a system design is to define how a system will behave. Notations for recording actual or potential system

behaviour have not attracted levels of interest comparable to those for system design notations. This is probably because detailed records of the behaviour of complex systems are mainly used out of sight within tools for system visualisation, verification, synthesis and failure analysis, rather than in documents and user interfaces.

The background of SO-nets and all their abstractions are in Section 2.2. In addition, we give an overview of time simulations and their techniques in Section 2.3. Finally, in Section 2.4 we provide a background of time granularity and their properties of layers.

2.2 Structured occurrence nets

2.2.1 Occurrence nets (O-nets)

Initially, occurrence nets (O-nets) were presented as representations of the execution processes in Petri nets [22]. Each process clearly and explicitly defines the relationships of concurrency and causality between performed events. Specifically, causally dependent occurrences of events are ordered, whereas their simultaneous occurrences are unordered. An O-net is a direct representation of the execution history of a system [21]. In addition to computer components, this system may also comprise systems and components that involve physical processes and individuals, such as those utilised in criminal and accident investigations. O-nets represent single executions of computing systems and the detail of the concurrency and causality relations between executed events [46].

An extension of the O-net formalism, the structured occurrence net (also known as SO-net) formalism was created to describe the behaviour of complex evolving systems, and is useful in sophisticated investigations. SO-nets are sets of related O-nets, utilising various types of clearly defined relationships and allowing different types of abstraction. Through these relationships, SO-nets can explicitly illustrate how systems evolve through different forms of communication, upgrades, reconfigurations, and replacements, and how one might take advantage from a complicated evolving system's behavioural knowledge [46]. The purpose of SO-nets is to express dependencies between interacting and evolving systems by

combining multiple related O-nets through the use of various formal relationships (specifically abstractions). The following sections presents all the abstractions within structured O-nets.

In [49] an O-net is a triple $onet = (C, E, F)$, where C and E are finite disjoint sets of *conditions* and *events*, respectively (referred to as the nodes), and $F \subseteq (C \times E) \cup (E \times C)$ is the *flow relation*. The *inputs* and *outputs* of a node x are ${}^\bullet x = \{y \mid (y,x) \in F\}$ and $x^\bullet = \{y \mid (x,y) \in F\}$. For $X \subseteq C \cup E$, ${}^\bullet X$ and $X^\bullet$ are the sets of all inputs and outputs of the nodes in X. Also, we have:

- For all $c \in C$: $|{}^\bullet c| \leq 1$ and $|c^\bullet| \leq 1$, and,
- For all $e \in E$: $|{}^\bullet e| \geq 1$ and $|e^\bullet| \geq 1$.
- The *causality relation* $\prec$ over E is acyclic, where $e \prec f$ if $e^\bullet \cap {}^\bullet f \neq \varnothing$.

In an O-net *onet*, a *marking* is any set of conditions. The initial and final markings are $M_0^{onet} = \{c \in C \mid {}^\bullet c = \varnothing\}$ and $M_{fin}^{onet} = \{c \in C \mid c^\bullet = \varnothing\}$.

A *step* of *onet* is a set of transitions U.

A *cut* of *onet* is a maximal set of conditions D such that $(c,d) \notin F^+$, for all distinct conditions $c, d \in D$.

An O-net can be executed by firing sets of events; this execution follows the rules of Petri net step semantics.

Definition 2.2.1 (O-net firing rule [49]) *Let $onet = (C, E, F)$ be an O-net, M be a marking, and U be a step.*

1. *U is (O-net)-enabled at M if ${}^\bullet U \subseteq M$.*
2. *If U is (O-net)-enabled at M, then U can be fired and create a new marking M' given by*

$$M' = (M \setminus {}^\bullet U) \cup U^\bullet.$$

This is indicated by $M[U\rangle_{onet} M'$.

3. *A* step sequence *of onet is a sequence* $\lambda = U_1 \dots U_n (n \geqslant 0)$ *of steps such that marking* $M_1 \dots M_n$ *exists, satisfying:*

$$M_0^{onet}[U_1\rangle_{onet} M_1 \dots M_{n-1}[U_n\rangle_{onet} M_n.$$

Figure 2.1 displays an O-net *onet* where circles indicate conditions and boxes represent events. The initial marking, $\{c_0\}$, is represented by the presence of a token within the starting condition. A possible step sequence is $\lambda = \{e_0\}\{e_1, e_2\}\{e_3\}$. The corresponding sequence of markings begins with $M_0^{onet} = \{c_0\}$ and ends with $M_{fin}^{onet} = \{c_5\}$. There are five cuts in the O-net of Figure 2.1: $\{c_0\}$, $\{c_1, c_2\}$, $\{c_1, c_4\}$, $\{c_3, c_4\}$, $\{c_2, c_3\}$, and $\{c_5\}$.

As defined in [49], a phase of O-net is a non-empty set of conditions, and each phase is a fragment of an O-net beginning with a cut and ending with a cut (a maximal set of causally unrelated conditions) that follows it in the causal sense, including all the conditions occurring between these cuts. A phase decomposition is a sequence of phases from the initial state to the final state, and whenever one phase ends, its maximal cut is the starting point of the successive one (minimal cut).

Formally, a *phase* of an O-net *onet* is a non-empty set of conditions $\pi \subseteq C$ such that the set $Min_\pi \subseteq \pi$ of the minimal conditions of π w.r.t. F^+ is a cut; the set $Max_\pi \subseteq \pi$ of the maximal conditions of π w.r.t. F^+ is a cut; and π comprises all conditions $c \in C$ for which there are $b \in Min_\pi$ and $d \in Max_\pi$ satisfying $(b, c) \in F^*$ and $(c, d) \in F^*$. Furthermore, a *phase decomposition* of *onet* is a sequence $\pi_1 \dots \pi_m$ of phases such that $M_0^{onet} = Min_{\pi_1}$, $Max_{\pi_i} = Min_{\pi_{i+1}}$ (for $i \leq m-1$), and $Max_{\pi_m} = M_{fin}^{onet}$.

We also say that an O-net *onet* is *line-like* if $C = \{c_1, \dots, c_l\}$, $E = \{e_1, \dots, e_{l-1}\}$, and $F = \{(c_1, e_1), (e_1, c_2), \dots, (c_{l-1}, e_{l-1}), (e_{l-1}, c_l)\}$. Such an O-net can be represented by the sequence $\xi_{onet} = c_1 e_1 \dots e_{l-1} c_l$.

2.2.2 Communication structured occurrence nets (CSO-nets)

In [49], a communication structured occurrence net (or CSO-net) was introduced to represent communication between different subsystems. A CSO-net is a set of O-nets that is commu-

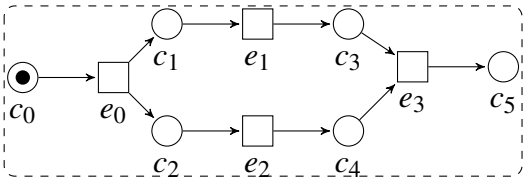


Figure 2.1 An occurrence net (O-net).

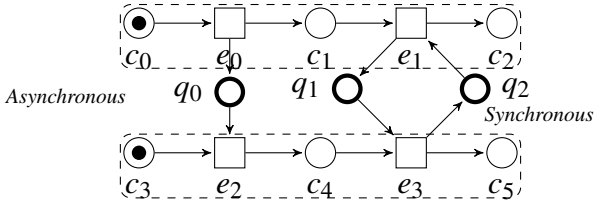


Figure 2.2 CSO-net with two interacting O-nets.

nicate *synchronously* or *asynchronously*, connected by special nodes called *channel places* (represented visually by bold circles). CSO-net can represent various types of communication among distinct systems. If an occurrence net is used to depict the collective behaviour of several interacting systems, it is useful to divide the model into individual component O-nets and design dedicated devices to symbolise communication among these subsystems. Figure 2.2 shows a CSO-net with two O-nets communicating *synchronously* (events e_1 and e_3 are always executed simultaneously) and *asynchronously* (events e_0 and e_2 can be executed simultaneously, or e_2 can be executed after e_0).

Definition 2.2.2 (CSO-net [49]) *A* CSO*-net is a tuple* $cson = (onet_1, \ldots, onet_k, Q, W)$*, where each* $onet_i = (C_i, E_i, F_i)$ *is an* O*-net (we denote* $\mathbf{C} = \bigcup_{i=1}^{k} C_i$*,* $\mathbf{E} = \bigcup_{i=1}^{k} E_i$ *and* $\mathbf{F} = \bigcup_{i=1}^{k} F_i$*),* Q *is a set of channel places, and* $W \subseteq (Q \times \mathbf{E}) \cup (\mathbf{E} \times Q)$ *are the arcs between the events and channel places. It is assumed that:*

1. *The* $onet_i$*'s and* Q *are mutually disjointed.*

2. *For every* $q \in Q$*, the inputs and outputs belong to distinct component* $onet_i$*'s,* $|{}^\bullet q| = 1$*, and* $|q^\bullet| \leq 1$*, where* ${}^\bullet q = \{e \in \mathbf{E} \mid (e,q) \in W\}$ *and* $q^\bullet = \{e \in \mathbf{E} \mid (q,e) \in W\}$*.*

3. *The relation* $(\sqsubset \cup \prec)^* \circ \prec \circ (\prec \cup \sqsubset)^*$ *over* $\mathbf{E}$ *is irreflexive, where:*

- $e \prec f$ *if there is* $c \in \mathbf{C}$ *with* $c \in e^\bullet \cap {}^\bullet f$*;*

- $e \sqsubset f$ *if there is* $q \in Q$ *with* $q \in e^{\bullet} \cap {}^{\bullet}f$. ♢

Intuitively, the original causality relation $\prec$ signifies the temporal order of events as "earlier than", while $\sqsubset$ indicates the relationship of events as "not later than". The inputs and outputs of a node in a CSO-net have been extended to include channel places.

A marking of *cson* is a set of conditions and channel places. The initial marking of a CSO-net is $M_0^{cson} = M_0^{onet_1} \cup \cdots \cup M_0^{onet_k}$, and the final marking is $M_{fin}^{cson} = M_{fin}^{onet_1} \cup \cdots \cup M_{fin}^{onet_k}$. In CSO-net, a step is a set of events that may come from one of more component O-nets.

Definition 2.2.3 (CSO-net firing rule [49]) *Let* CSO*-net be as in Definition 2.2.2, M be a marking, and U be a step of the* CSO*-net.*

1. *U is (*CSO*-net)-enabled at M if* $({}^{\bullet}U \setminus (U^{\bullet} \cap Q)) \subseteq M$.

2. *If U is (*CSO*-net)-enabled at M, then U can be fired and create a new marking* $M' = (M \cup U^{\bullet}) \setminus {}^{\bullet}U$. *This is indicated by* $M[U\rangle_{cson} M'$. ♢

The *step sequences* and *reachable markings* of CSO-net are then described in a similar way as for an O-net.

The firing rule in Definition 2.2.3 a step U involving synchronous behaviour is permitted to utilise both the tokens that are currently present in "channel places" at marking M and the tokens placed there by actions taken in step U when U was being executed. Consequently, elements originating from step U are capable of mutually assisting one another individually and moving resources (tokens) synchronously. Therefore, unlike the step sequence in O-net, where each step is composed of multiple enabled events, the execution of a step in CSO-net (i.e. $M[U\rangle M'$.) might involve *synchronous* communications, which refer to the behaviour of events executing concurrently and behave as a single transaction. This mode of execution permits the simultaneous execution of multiple events, rendering it more expressive in comparison to the one employed in O-nets.

2.2.3 Behavioural structured occurrence nets (BSO-nets)

Behavioural structured occurrence nets (or BSO-nets) model activities of evolving systems [49]. An execution history is represented on two different levels: the lower level, which is used to indicate behavioural details, and the upper level, which is used to represent the stages (phases) of system evolution. Thus, a BSO-net provides details about the evolution of a system.

Definition 2.2.4 (BSO-net [49]) *Let* CSO*-net be a communication structured occurrence net as in Definition 2.2.2, and* $cson^{\uparrow} = (onet_1^{\uparrow}, \ldots, onet_m^{\uparrow}, Q^{\uparrow}, W^{\uparrow})$ *be a disjoint* CSO*-net such that* $onet_i^{\uparrow} = (C_i^{\uparrow}, E_i^{\uparrow}, F_i^{\uparrow})$ *is line-like, for* $i \leq m$*. In addition, let* $\mathbf{C}^{\uparrow} = \bigcup_{i=1}^{m} C_i^{\uparrow}, \mathbf{E}^{\uparrow} = \bigcup_{i=1}^{m} E_i^{\uparrow}$*, and* $\mathbf{F}^{\uparrow} = \bigcup_{i=1}^{m} F_i^{\uparrow}$*.*

A behavioural structured occurrence net (or BSO*-net) is a tuple*

$$bson = (cson, cson^{\uparrow}, \beta).$$

such that $\beta \subseteq \mathbf{C} \times \mathbf{C}^{\uparrow}$*.*

It is assumed that the following hold:

1. *For every* $onet_i$*, there exists exactly one* $onet_j^{\uparrow}$ *satisfying* $\beta(C_i) \cap C_j^{\uparrow} \neq \varnothing$*.*
2. *For every* $onet_j^{\uparrow}$ *represented by a chain* $\xi_{onet_j^{\uparrow}} = c_1 e_1 \ldots e_{l-1} c_l$*, the sequence* $\pi_1 \pi_2 \ldots \pi_l = \beta^{-}1(c_1)\beta^{-}1(c_2)\ldots\beta^{-}1(c_l)$ *is a concatenation of phase decompositions of different* O*-nets in cson. For all* c_j *and* e_j *occurring in the chain* $\xi_{onet_i^{\uparrow}}$*, we denote* $\pi(c_j) = \pi_j$*, and* $causal(e_j) = (pre(Max_{\beta-1(pre(e_j))}) \times \{e_j\}) \cup (\{e_j\} \times post(Min_{\beta-1(post(e_j))}))$*.*
3. *The relation*

$$(\sqsubset \cup \prec \cup \lhd)^* \circ (\prec \cup \lhd) \circ (\prec \cup \sqsubset \cup \lhd)^*$$

over $\mathbf{E} \cup \mathbf{E}^{\uparrow}$ *is irreflexive, where:*

- $e \prec f$ *if there is* $c \in \mathbf{C} \cup \mathbf{C}^{\uparrow}$ *with* $c \in e^{\bullet} \cap {}^{\bullet}f$*;*
- $e \sqsubset f$ *if there is* $q \in Q \cup Q^{\uparrow}$ *with* $q \in e^{\bullet} \cap {}^{\bullet}f$*; and*

- $e \lhd f$ *if* $(e,f) \in \bigcup_{e' \in \mathbf{E}^{\uparrow}} causal(e')$.

The initial marking M_0^{bson} *of* BSO*-net is the initial marking of* $cson^{\uparrow}$ *together with the initial markings of all the* $onet_i$*'s such that* $\beta(M_0^{onet_i}) \cap M_0^{cson^{\uparrow}} \neq \varnothing$. *The final marking* M_{fin}^{bson} *of the* BSO*-net is the final marking of* $cson^{\uparrow}$ *together with the final markings of all the* $onet_i$*'s such that* $\beta(M_{fin}^{onet_i}) \cap M_{fin}^{cson^{\uparrow}} \neq \varnothing$. ♢

A BSO-net consists of two CSO-nets linked by a behavioural relation β. Definition 2.2.4(1) states that each phase corresponds to one condition of the upper level O-net, whereas Definition 2.2.4(2) indicates that each upper-level condition is linked to a single phase of a lower level O-net. The sequence of the upper-level conditions must match to the phase decompositions of the lower-level O-net. Definition 2.2.4(3) specifies that the new dependencies, along with the existing communication ($\sqsubset$) and causal relations ($\prec$) in the model, must be acyclic.

Figure 2.3 shows an example of BSO-net representing a system update. The model illustrates a version change brought on by an update event represented at the higher level. The lower level reveals the behaviour of the system before and after the update. The dashed lines utilised in this context serve to depict the pertinent connections between the two levels of behaviours [49].

Definition 2.2.5 (BSO-net firing rule [49]) *Let* BSO*-net be as in Definition 2.2.4,* $M \subseteq \mathbf{C} \cup \mathbf{C}^{\uparrow}$ *be a marking and* $U \subseteq \mathbf{E} \cup \mathbf{E}^{\uparrow}$ *be a step of* BSO*-net.*

1. *U is (*BSO*-net)-enabled at M if*

   - $({}^{\bullet}U \setminus U^{\bullet}) \subseteq M$;
   - $Max_{\pi({}^{\bullet}e)} \subseteq M$, *for each* $e \in \mathbf{E}^{\uparrow} \cap U$;
   - $\beta({}^{\bullet}e') \cap \beta(e'^{\bullet}) \in M$, *for each* $e' \in \mathbf{E} \cap U$.

2. *If U is (*BSO*-net)-enabled at M, then U can be fired and create a marking* M' *given by:*

$$M' = (M \setminus ({}^{\bullet}U \cup Max_{\pi({}^{\bullet}U)})) \cup U^{\bullet} \cup Min_{\pi(U^{\bullet})}$$

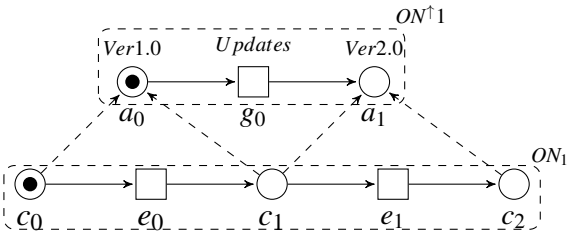


Figure 2.3 BSO-net portraying a system update

where $Max_{\pi(\bullet U)} = \bigcup_{e \in U} Max_{\pi(\bullet e)}$ *and* $Min_{\pi(U\bullet)} = \bigcup_{e \in U} Min_{\pi(e\bullet)}$.
This is denoted by $M[U\rangle_{bson} M'$. ♢

The firing rule mentioned above manages the marking movement between different phases. To determine if a step is (BSO-net)-enabled based on a marking, three requirements must be met: (i) the system is enabled by (CSO-net) (Definition 2.2.3); (ii) The maximal conditions in the phase of the input condition for each higher-level event in U are present in the current marking. (iii) Similarly, for each lower-level event in U, the corresponding higher-level condition is present in the current marking.

For the BSO-net in Figure 2.3, $\{e_0\}\{g_0\}\{e_1\}$ is a possible step sequence. The only step U_1 enabled at the initial marking $\{a_0, c_0\}$ is $U_1 = \{e_0\}$. The firing of U_1 changes the marking to $\{a_0, c_1\}$ which enables the step $U_2 = \{g_0\}$. The firing of U_2 produces $\{a_1, c_1\}$ and also enables $U_3 = \{e_1\}$ which produces the final marking $\{a_1, c_2\}$.

2.2.4 Alternative occurrence nets (ALTO-nets)

In [49], an extension of SO-nets called alternative structured occurrence nets (ALTO-nets) was introduced to facilitate the modelling of alternative behaviours. It can be used to simulate and analyse increasingly complicated dynamic systems, including (cyber) crimes or accidents. The concept of incorporating alternatives to SO-nets initially formed from [47] in order to analyse and simulate increasingly more complex evolving systems.

Definition 2.2.6 (TAGO-net [49]) *Let* $AS = \{A_1, \ldots, A_\varphi\}$ *be a set (of alternative scenarios). A* tagged O-net *(or* TAGO*-net) is a tuple tagon* $= (C, E, F, \vartheta)$*, where: C and E are disjoint sets of conditions and events (collectively referred to as the nodes), respectively;* $F \subseteq$

$(C \times E) \cup (E \times C)$ *is the flow relation; and* $\vartheta : C \cup E \cup F \to 2^{AS} \setminus \{\varnothing\}$ *is a mapping, such that, for each* $A \in AS$,

$$tagon(A) = (C(A), E(A), F(A))$$

is an O*-net, where for* $X \in \{C, E, F\}, X(A)$ *is the set of all* $x \in X$ *with* A *appearing in* $\vartheta(x)$.

The initial marking M_0^{tagon}*, final marking* M_{fin}^{tagon}*, input and output of node* x *(i.e.,* ${}^\bullet x$ *and* $x^\bullet$*), in tagon are defined in the same way as for* O*-nets. It is further assumed that for all* $A \in AS$, $M_0^{tagon} = M_0^{tagon(A)}$ *and* $M_{fin}^{tagon} = M_{fin}^{tagon(A)}$. ♢

Two nodes, denoted as x and y, are considered to be *alternatively related (or conflicting)* if there exist separate events, e and f, such that ${}^\bullet e \cap {}^\bullet f \neq \varnothing$ and $(e,x) \in F^*$ and $(f,y) \in F^*$ (denoted by $x\#y$). They are causally related if $(x,y) \in F^+$ or $(y,x) \in F^+$. A *block* of *tagon* is a non-empty set $B \subseteq (C \cup E)$ such that $B \cap C = {}^\bullet(B \cap E) \cap (B \cap E)^\bullet$, $({}^\bullet B \setminus B) \times (B^\bullet \setminus B) \subseteq \prec$, and for all $e, f \in (B \cap E)$, $\neg(e\#f)$.

Definition 2.2.6 encapsulates the fundamental concept of illustrating the different behaviours of the system. A *tagon* can be seen as a combination of several O-nets, every labelled with a symbol A in AS that represents a distinct scenario (or world). This is determined by the mapping ϑ. Therefore, to identify which scenarios an element belongs to, each element in a *tagon* is labelled by one or more tags. The tagging is not random; items with the same tag create a valid O-nets (i.e. *tagon*(A)). The behaviour report of what has occurred is presented, and it is viewed from the perspective of one of the possible scenarios $A \in AS$.

The example shows in Figure 2.4 that, by applying the concept proposed by the SO-net in Figure 2.4(a), it would be necessary to depict the various routes taken by an individual while travelling to a specific location on distinct ones (a and b), with each one symbolising a distinct scenario. This method of modelling generates numerous duplicate states, as these

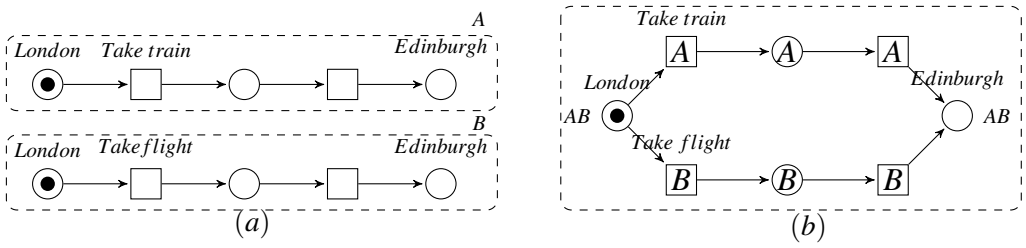


Figure 2.4 (a) Scenarios modelled by two O-nets; and (b) ALTO-net

two ones describe the exact same system. As illustrated in Figure 2.4(b), ALTO-nets are enhanced to account for this circumstance. By 'glueing' together common states (conditions), the ones in (a) are essentially integrated into a single structure as in (b).

Definition 2.2.7 (ALTO-net [49]) *Let* $tagon^{\downarrow} = (C^{\downarrow}, E^{\downarrow}, F^{\downarrow}, \vartheta^{\downarrow})$ *be a* TAGO*-net as in Definition 2.2.6, and* $tagon = (C, E, F, \vartheta)$ *be a sequential* TAGO*-net (this means that* $|M_0^{tagon}| = 1$ *and* $|{}^{\bullet}e| = |e^{\bullet}| = 1$*, for every* $e \in E$*).*
An alternative O-net *(or* ALTO*-net) is a pair*

$$alton = (tagon^{\downarrow}, \tau)$$

such that $\tau : C^{\downarrow} \cup E^{\downarrow} \to C \cup E$ *is a mapping from the nodes of* $tagon^{\downarrow}$ *to the nodes of tagon. Assumptions are made regarding the following:*

1. $\tau(C^{\downarrow} \cup E^{\downarrow}) = C \cup E, \tau^{-1}(C) \subseteq C^{\downarrow}$*, and* $\tau^{-1}(E) = E^{\downarrow}$.
2. *For all* $e \in E, \tau^{-1}(e)$ *are disjoint blocks of* $tagon^{\downarrow}$.
3. *For all* $c \in C, |\tau(c)| = 1$.
4. $F = \{(x,y) \in (C \times E) \cup (E \times C) | (\tau^{-1}(x) \times \tau^{-1}(y)) \cap F^{\downarrow} \neq \varnothing\}$.
5. *For all* $x \in (C^{\downarrow} \cup E^{\downarrow} \cup F^{\downarrow}), \vartheta(x) = \vartheta(\tau(x))$. ♢

Definition 2.2.8 (ALTO-nets firing rule [49]) *Let alton be an* ALTO*-net, M be a marking and U be a step of alton, and* $A \in AS$.

1. *U is (*ALTO*-net)-enabled at M and A if the following hold:*
   - $M(c) \geq \sum_{e \in c^{\bullet}} U(e)$*, for every condition* $c \in C$*; and*
   - $A \in \vartheta(e)$*, for every event* $e \in U$.
2. *If U is (*ALTO*-net)-enabled at M and A, then U can be fired and create a new marking* $M' = (M \setminus {}^{\bullet}U) \cup U^{\bullet}$*. This is denoted by* $M[U\rangle^{A}_{alton} M'$.

A step sequence of alton (w.r.t. scenario A) is a sequence $U_1 \ldots U_n (n \geq 0)$ *of steps, such that there exist markings* $M_1, \ldots, M_n$ *satisfying:*

$$M_0^{alton}[U_1\rangle_{alton}^{A} M_1, \ldots, M_{n-1}[U_n\rangle_{alton}^{A} M_n.$$

Thus a step sequence portrays a valid system execution with respect to a particular scenario.

The Definision 2.2.8 above describes the ALTO-net firing rule. Using different scenarios, it shows how the states of a system change. Specifically, if a marked situation leads to more than one event, only one of them can be selected to fire. Note that in the ALTO-net, a step is a set of events.

2.2.5 Time in SO-nets

In [23], timed SO-nets are based on groups of associated timed occurrence nets and are designed for reasoning about related events and causality with time information that is uncertain or missing. When modelling a system based on time, such as accidents or crimes, it is important to identify/estimate the order in which events have happened and to identify the duration of the events. In [23] each node in the SO-net (condition, event, and channel place) has a start time (T_s), finish time (T_f) and duration (D). As shown in Figure 2.5, all time values have bounded uncertainty represented via specified times intervals $([T_{s,e}, T_{s,l}])$ which mean (start lower and upper respectively) and $([T_{f,e}, T_{f,l}])$ which mean (finish lower and upper respectively). In addition, the duration has bounded uncertainty represented via a duration interval $([D_e, D_l])$ (duration lower and upper).

As shown in Figure 2.6, the time interval information is specified on each arc (prefixed by 'T :'). The time information represents the finish time of the source of the node in addition to the start time of the destination node. The duration interval (is prefixed by 'D :') [49].

To summarise, in this section we introduced several kinds of nets which can be used to represent behaviours of concurrent systems:

- Occurrence nets provide basic behaviour representations of concurrent system behaviours. Each occurrence net is an acyclic net representing in an explicit way the

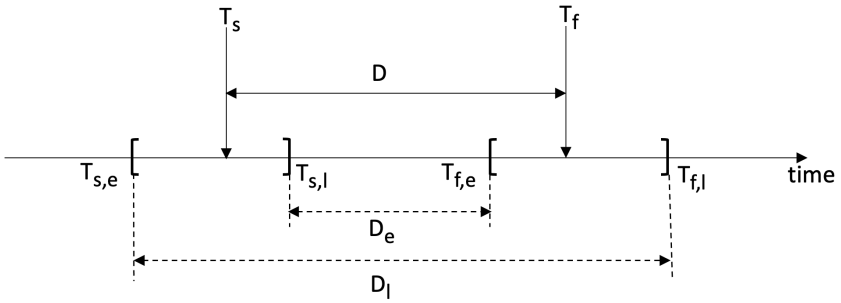


Figure 2.5 Relationships between unknown and known time values and duration of a node on a global timeline

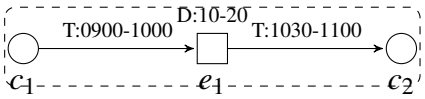


Figure 2.6 Time information of O-nets

concurrency and causality relationships between events in a single run of a concurrent system.

- Communication structured occurrence nets (CSO-nets) provide behaviour representations of execution runs involving several communicating concurrent systems. These communications can be synchronous or asynchronous, and they are made possible by special channel places. Each CSO-net represents a single execution run of a system.

- Behavioural structured occurrence nets (BSO-nets) provide a means of capturing abstraction in behavioural descriptions expressed using CSO-nets. Each BSO-nets relates two such descriptions: one specifying the detailed behaviour, and the other specifying the same behaviour using, for example, single events instead of sequences of executed events.

- Alternative occurrence nets (ALTO-nets) differ from the previous three kinds of nets by allowing one to specify several alternative (related) behaviours – captured by CSO-nets - within a single structure.

We have also indicated how to annotate the above nets in order to provide timing information related to the events involved in a behavioural description.

2.3 Time simulation

In this section, we concentrate on the history of time simulation as well as its significance. In addition, we examine the methodologies and techniques of time simulation.

Since the middle of the twentieth century, simulation tools have been widely used for the design and development of electrical systems [18]. Time simulation is also a powerful tool used in many fields to model and analyse the behaviour of complex systems over time, such as crimes and accidents. It can be used in a wide range of fields, including engineering, physics, economics, and biology. For example, engineers might use time simulation to model the behaviour of a complex mechanical system, such as an airplane or a car, to test its performance under different conditions. Similarly, physicists might use time simulation to study the behaviour of particles in a particular environment, while economists might use it to model the behaviour of financial markets over time. In [16], simulation is a crucial problem-solving methodology that plays an integral role in addressing a wide range of real-world situations. Simulation employed to explain and evaluate the dynamics of a given system, enabling the formulation of hypothetical scenarios and facilitating the development of real systems, and it can be used to model both existent and conceptual systems. A simulation makes a model come to life and demonstrates the behaviour of a certain object or phenomenon. A simulation also can be helpful for teaching, analysis, or testing when real-world systems or concepts can be modelled [33]. In the section that follows, we briefly explain the importance and techniques of time simulation.

2.3.1 Relevance of time simulation

Shannon (1975) defined simulation as

> 'the process of designing a model of a real system and conducting experiments with this model for the purpose of either understanding the behaviour of the system or evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system' [42].

Simulations of time can be beneficial for researchers in a variety of disciplines, allowing them to comprehend the sequence of events, reconstruct scenarios, and analyse the dynamics of a situation over time. Simulation models can be used to study how complicated processes behave [24]. They can aid in clarifying assumptions, also known as mental models, regarding the operation of a process [64]. Time simulation is an effective technique for estimating the performance of novel systems prior to their construction, and can be used to enhance the efficacy of the existing system. Simulating a modelled system can aid in the design of future systems and the enhancement of extant ones. Additionally, time simulation enables the exploration of a real system by modifying policies, operations, and methods at a relatively low-cost, without causing interference with the actual system; and speed up or slow down an interesting event so that it can be studied in more detail [41].

2.3.2 Time simulation techniques

In recent decades, various time simulation techniques have been elaborated. As a result, the functionality of a simulation model is contingent on the chosen modelling technique. There are numerous event-driven simulation techniques available, and the manner in which a simulation model operates is contingent upon the modelling methodology selected. There are, in general, four significant differentiations that can be drawn between various simulation methodologies. There are many time simulation techniques, but the most commonly employed is discrete event simulation. In the section that follows, we briefly explain time simulation techniques.

2.3.2.1 Discrete event simulation

In this section, we discuss the discrete event simulation models that have been widely employed in academia and industry to address a wide range of industrial issues [13]. Discrete event simulation is considered to be one of the most significant types of simulation; it represents the system as a sequential progression of discrete events that occur over time, and aims to represent the actions that the entities take to furnish insight into the dynamic behaviour of the system [73]. Discrete event simulation is a useful approach for addressing

challenges characterised by variables that undergo changes at specific points in time and in discrete increments [61]. Moreover, a discrete event simulation is executed (or 'run') over time by a mechanism that advances simulated time. At each event, the system state is updated, along with any resources that may be captured or released [15].

2.3.2.2 Deterministic and stochastic simulation

Deterministic computer models are used in numerous different areas of science, engineering, and policy making. Usually, they are complicated models that try to capture the underlying processes in great detail, and they have many of inputs that the user specifies [66]. For a given set of input parameter values in a deterministic simulation model, the output parameter values produced during simulation runs will always be the same [64]. To run deterministic simulations, we need to have accurate models and an in-depth knowledge of the dynamics and interactions of the system. Stochastic simulation models have elements of probability. In a stochastic simulation model, the values of the output parameters can change based on how the values of the input parameters or intermediate (internal) model factors change. Since the changes in input and intermediate variables are caused by random picking from known statistical distributions, it is important to run the stochastic simulation enough times to see the statistical distribution of the values of the output parameters [64].

2.3.2.3 Static and dynamic simulation

Static simulation models are designed to represent the variability of model parameters at a specific point in time. It is a useful method for examining data, systems, and processes at a particular moment in time, providing insights into their composition, characteristics, and possible problems. It is especially helpful in situations where the analysis's main focus is not on the system's dynamic evolution [64]. Dynamic simulation models are able to capture the behaviour of model parameters over a predetermined amount of time [64]. As a result, for predicting the behaviour of a system under the influence of different policies of interest, dynamic simulation models are particularly valuable. By providing a consistent foundation

for the predictions, dynamic simulation models offer substantial benefits when employed in this capacity[74].

2.3.2.4 Continuous simulation

Continuous simulation is an appropriate approach for systems characterised by continuous changes in variables [61]. It is distinguished from discrete event simulation by the occurrence of changes at specific, separate time intervals. Moreover, continuous simulation models involve the iterative modification of model variables, which describe the state of the model, at regular intervals of time. This process is guided by a predetermined set of well-specified model equations. In continuous simulation models, the model equations serve to build a system of time-dependent linear differential equations, which might be of first or higher-order [64].

2.3.2.5 Quantitative and qualitative simulation

In this part, we provide an overview of quantitative and qualitative simulation. To conduct quantitative simulation, it is necessary to define the values of model parameters as either real or integer numbers. Therefore, one of the most important requirements for quantitative simulation is the availability of either a sufficient quantity and quality of empirical data or the availability of specialists who are ready to estimate model parameters quantitatively. It is helpful to use qualitative simulations when trying to understand how dynamic systems usually act or when there is not enough data to draw specific conclusions. [64].

2.3.2.6 Hybrid simulation

Hybrid simulation is an approach that combines different simulation techniques to analyse and understand complex systems or processes [76]. Hybrid simulation models are dynamical planning tools that combine parts of being continuous with event-driven or deterministic with stochastic elements. One advantage of hybrid techniques is in their potential to integrate the benefits offered by stochastic, continuous, and event-driven models. As a result, it provides a

more accurate representation of the real-world behaviour of complex systems by combining the strengths of simulation modelling techniques [64].

2.4 Time granularity

In this section, we concentrate on the history of time granularity. A fundamental issue in every computer system that involves representing the world is the representation of time. Applications including databases, simulation, expert systems, and general applications of artificial intelligence fall under this category [9].

Time granularity refers to the level of detail or precision at which time is measured or represented in various contexts, such as data analysis, measurement, scheduling, and technology. It is an essential concept in fields like computer science, physics, data science, and everyday life. Time granularity can vary widely depending on the specific application or requirements. It is important for a wide range of scientific and industrial processes; therefore, time granularity is an important part of knowing how actions that happen at coarse levels of time interact with others that happen with more details at finer levels. In addition, in temporal reasoning systems, time is typically viewed as a linear sequence of discrete points or linear intervals. Intervals are grouped into what are known as convex intervals to reflect various levels of time granularity. The granularity is crucial when examining the world from various levels of abstraction, and when transitioning between these levels, it is essential for understanding the phenomena under observation [58]. In the present section we give an overview of the properties of layers for the time granularity.

In [32] a temporal scenario can be described using various levels of abstraction, which hinges on the needed precision or the extent of available knowledge. Time granularity, in this context, represents the ability to specify the temporal details of a statement. The introduction of a formal framework incorporating the notion of time granularity facilitates the modelling of temporal information across various levels of temporal precision.

2.4.1 Relevance of time granularity

The level of time granularity is critical when simulating complex systems, as it enables precise and adaptable representation of the dynamics of these intricate systems. Complex systems frequently comprise a multitude of interdependent elements functioning at varying temporal dimensions [63]. Through the manipulation of time granularity, modellers are able to concentrate on particular facets or expand their attention to examine the overall behaviour of the system [32]. Coarse-grained time granularity can be perceived as a limited number of components, that perform high-level actions or represent complex elements, whereas fine-grained time granularity facilitates the capture of local interactions and rapid fluctuations. Modellers are able to create more realistic and effective representations of complex systems through the ability to toggle between these levels of temporal detail. This capability promotes improved decision-making and problem-solving [31]. Choosing the appropriate time granularity levels ensures that the model captures the relevant temporal aspects, leading to more reliable predictions and a deeper understanding of how complex systems function and evolve over time.

2.4.2 Properties of Layers

Layers in time granularity are defined by their hierarchical structure, whether they overlap or are disjoint, and the amount of detail they provide when showing time intervals. Thus, layers might overlap, such as Days and Working Days, where every working day is a day, or be disjoint, like Days and Weeks. Understanding this distinction is essential for grasping the hierarchical structure of time intervals and their relationships within a temporal framework [32].

Take into consideration the scenario in which a situation is defined concerning the completely ordered collection of granularities, which includes years, months, weeks, and days. The relationships that exist between these levels are different. The following concepts can be used to describe the distinctions:

Homogeneity: It means that the temporal of entities in the higher layer is equal to the number of entities in the lower layer;

Alignment: when only one of the entities in the coarser layer maps to entities in the lower layer.

The above two concepts enable us to identify the following distinct cases:

Year-Month: Each year comprises an identical number of months (homogeneity), and each month is linked to a single year (alignment); thus, the relationship between years and months is quite straightforward.

Year-Week: A year might have a varying number of weeks (non-homogeneity), and a week can be associated with more than one year (non-alignment).

Month-Day: Each day is precisely linked to a single month (alignment), and the number of days within a month can change (non-homogeneity).

2.5 Conclusion

In this chapter, we have examined and discussed various background areas. An overview of SO-nets and their features has been provided, along with their definitions. We provided an overview of CSO-nets and offered examples illustrating how these communications occur, along with an overview of BSO-nets, which are employed to model the activities of evolving systems. These nets are represented in two different models.

Additionally, we have discussed ALTO-nets, introduced to facilitate the modelling of alternative behaviours. We have provided a background of time simulation and reviewed its techniques to show how to simulate the behaviour of the system using time. Several studies have explored the comparison between continuous simulation and discrete event simulation, with detailed discussions available in the time simulation Section 4.4 (Chapter 4). We also provided an overview of time granularity and its significance in simulating complex systems across various levels of time units. Also, we have discussed the properties layers of hierarchical structure of time granularity. We analysed the hierarchical structure of temporal

granularity, emphasising its different levels and characteristics. We also discussed these layers at various scales.

Chapter 3

Consistency of Time Information

This chapter outlines our approach for dealing with timed SO-nets. In particular, we introduce algorithms for computing the missing time information and checking consistency.

3.1 Introduction

Time imparts valuable insights into behaviour and data. The notion of time is essential to our comprehension of events and processes. By including time-related characteristics in system models, they gain the ability to recognise and exploit the sequential progression of events, thereby facilitating a more appropriate comprehension of evolving phenomena. The incorporation of temporal data into models spanning multiple domains has emerged as a critical component in the progression of predictive analytics.

This chapter provides an introduction to the (*date-time*) property in relation to the fundamental concepts of SO-net and ALTO-net. Time representation is an appealing feature of modelling complex evolving systems. Many information systems designed in software engineering must deal with time data; for instance, the time sequence of events is an important part of many applications.

In a criminal investigation, putting together a timeline of a crime for each suspect helps the organisation of evidence into a coherent showing for a court of law. In extensively instances, however, the available temporal information regarding an event or condition is

imprecise or insufficient. For example, it might not be feasible to specify the precise time information at which a robbery occurred, but it might be feasible to specify intervals of times during which the robbery occurred.

Therefore, the contribution of this chapter is a novel tool-supported formalism (timed SO-nets) for modelling and reasoning about concurrent events with uncertain or missing time information in developing systems. It is built on collections of connected timed occurrence nets.

This chapter is divided into six main sections, Section 3.2 provides the notation for timed SO-nets, which are based on discrete time intervals. The conditions for verifying the consistency of (*date-time*) intervals are provided in Section 3.3. In Section 3.4, algorithms are given for estimating and enhancing the accuracy of *(date-time)* intervals. These algorithms utilise default duration intervals and redundant time information. The algorithm for estimating and improving the accuracy of *(date-time)* intervals for alternative occurrence nets is defined in Section 3.5. Algorithms for checking consistency in SO-nets and ALTO-net are given in Section 3.6, and the conclusion of the chapter is in Section 3.8.

3.2 Time model of SO-nets

A time model is an abstract structure or graphical representation that is specifically engineered to analyse the temporal characteristics of a given system. Computer science is one of the many disciplines that employs time models to comprehend and analyse the progression of events over time. In a time-based system, it is critical to establish the order in which events have fired and the duration of intervals between them to determine, e.g., eliminate improbable hypothesised scenarios.

The way in which time is represented (using dates, etc.) is much more practical for supporting, for example, crime investigations than number-based attempts in [23]. Representing (date-time) information about such systems is important. Timed SO-nets are based on groups of associated timed O-nets and are designed for reasoning about related events and causality with time information that is uncertain or missing in evolving systems. When modelling a

system based on time, such as accidents or crimes, it is important to identify the order in which events have happened and the duration of the events. However, the temporal information that is known about an occurrence is often inaccurate or lacking. For instance, although it may not be possible to pinpoint a certain date-time (such as [2022/11/04 05:05:05]) at which a robbery happened, it could be possible to provide temporal boundaries such as (earliest start and latest start). The notations and definitions in this chapter are adapted from [23].

We assume that each node (condition, event) in SO-nets has a start (date-time) (T_s) and finish (date-time) (T_f), and that each (date-time) event has a bounded uncertainty that is represented by a specified earliest and latest (date-time) interval $([T_{s,e}, T_{s,l}])$ and $([T_{f,e}, T_{f,l}]$ respectively). Furthermore, each node has a duration (D) with a bounded uncertainty represented by a specified early and late duration interval $([D_e, D_l])$, and a calculated earliest $(shortest)$ and latest $(longest)$ duration between the start and end (date-time) of each node $([D_e, D_l])$.

We now introduce formally some notations for time information related to the SO-nets framework. Let n be a node in a SO-nets. The early and late start (date-time) interval of n is denoted by:

$$I_s^n = [T_{s,e}^n, T_{s,l}^n]$$

where $T_{s,e}^n$ and $T_{s,l}^n$ are the earliest and latest start (date-time), respectively. I_s^n is considered well-defined only if the following inequality is met:

$$T_{s,e}^n \leq T_{s,l}^n \tag{3.1}$$

The early and late finish (date-time) interval of n is denoted by:

$$I_f^n = [T_{f,e}^n, T_{f,l}^n]$$

where $T_{f,e}^n$ and $T_{f,l}^n$ are the earliest and latest finish (date-time), respectively. I_f^n is considered well-defined only if the following inequality is met:

$$T_{f,e}^n \leq T_{f,l}^n \tag{3.2}$$

The constraint $(T_s^n) \leq (T_f^n)$ states that the start time of n must be at or earlier than the finish time of n. The start and finish time intervals of n should satisfy the following conditions in order to assure consistency with this constraint.

$$T_{s,e}^n \leq T_{f,e}^n \wedge T_{s,l}^n \leq T_{f,l}^n \tag{3.3}$$

The early and late duration interval of n are denoted by:

$$I_d^n = [D_e^n, D_l^n]$$

where D_e^n and D_l^n are the earliest and latest duration intervals. I_d^n is considered well-defined only if the following inequality is met:

$$0 \leq D_e^n \leq D_l^n \tag{3.4}$$

3.3 Time consistency

3.3.1 Time consistency in line-like O-nets

Time consistency in line-like O-nets, each event has exactly one input condition and one output condition, and each condition has at most one input and output event. Then, for any two directly connected nodes (i.e., a condition that ends in an event or an event that starts a condition), we assume that the finish (date-time) of the source node is equal to the start (date-time) of the destination node. Consequently, we have:

$$I_f^{n_1} = I_s^{n_2}, \quad \textit{for every} \ \ (n_1, n_2) \in F \tag{3.5}$$

Let n be a node in O-net. The information regarding the start, finish (date-time), and duration of n is defined to be *node consistent* only if the following inequalities are satisfied:

$$[T_{s,e} + D_e, T_{s,l} + D_l] \cap [T_{f,e}, T_{f,l}] \neq \varnothing \tag{3.6}$$

$$[T_{f,e} - D_l, T_{f,l} - D_e] \cap [T_{s,e}, T_{s,l}] \neq \varnothing \tag{3.7}$$

$$[max(0, T_{f,e} - T_{s,l}), T_{f,l} - T_{s,e}] \cap [D_e, D_l] \neq \varnothing \tag{3.8}$$

For example, the Eq.(3.6) verifies the bounds are consistent (i.e., overlap) with the specified finish (date-time) interval of n. Eq.(3.7) validates that the bounds are in accordance with the node's specified start (date-time) interval n. Eq.(3.8) verifies that the specified start and end (date-time) intervals of n are used to compute the bounds on uncertainty for the duration interval of n. A line-like O-net is time consistent if and only if every node n, in the O-net is node consistent and the flow relation F of the O-net satisfies Eq.(3.5).

For example, consider the line-like O-nets shown in Figure 3.1. The (date-time) intervals are shown above each node (with the prefixes *Estart* and *Lstart* representing the early and late start (date-time) intervals, respectively), and the early, late finish (date-time) intervals (with the prefixes *Efinish* and *Lfinish* (date-time) intervals). In addition, the duration interval of the event node is prefixed by *Eduration* and *Lduration* in the format

$$(Y{:}Year, M{:}Month, D{:}Day, H{:}Hour, Min{:}Minute, S{:}Second).$$

Using Eq.(3.6) above, it can be observed that the time information in event e_0 Figure 3.1 (a) is inconsistent. Its estimated finish time-interval is

$$[T_{s,e} + D_e, T_{s,l} + D_l] = [Efinish{:}2022/10/02\ 12{:}00{:}00, Lfinish{:}2022/10/02\ 14{:}00{:}00],$$

and its specified finish time-interval is

$$[Efinish{:}2022/10/02\ 13{:}00{:}00, Lfinish{:}2022/10/02\ 14{:}00{:}00].$$

In contrast, event e_0 in Figure 3.1 (b) is node consistent.

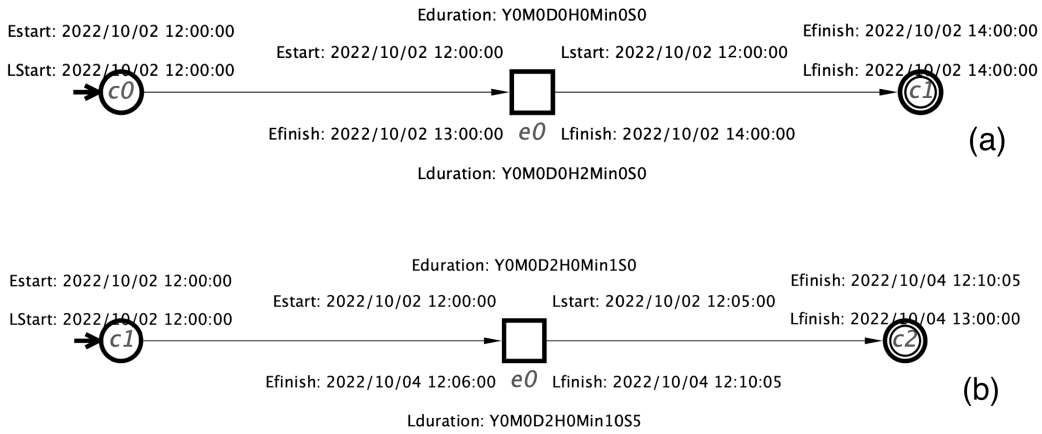


Figure 3.1 Line-like O-net with date and time interval.

3.3.2 Time consistency in CSO-nets

In [49], a communication structured occurrence net (CSO-net) captures communication between different subsystems. Communication between events is represented using a special nodes called *channel place* that behave similarly to conditions. In asynchronous communication, the sending event e, executes either before, or simultaneously with, the receiving event e', an *asynchronous channel place* connects the two events and uses a condition to record information about the communication. In synchronous communication involves the simultaneous execution of two communicating events, which are connected by two *synchronous channel places*. These channels record the communication information using conditions and have the same temporal characteristics as the events.

Formally, let q denote a *channel place*, and let e and e' represent the input and output events of q, respectively. The temporal information of q is considered *a/synchronously consistent* only if the following conditions are met:

$$I_f^e = I_s^q \tag{3.9}$$

$$I_s^{e'} = I_f^q \tag{3.10}$$

$$q \textit{ is node consistent} \tag{3.11}$$

Intuitively, Eq.(3.9) can be seen as stating that the start time interval of q equals to the end time interval of its input event. Eq.(3.10) asserts that the end time interval of q equals to the start time interval of its output event. According to Eq.(3.11), q must satisfy node consistency in relation to its three time and duration intervals. The consistency checking of an asynchronous channel place, denoted by q, involves verifying a condition that can persist for a non-zero duration. If q is a synchronous channel place, its duration is zero because of the cyclic representation of synchronous communication.

Figure 3.2 shows how (date-time) information in a CSO-nets can reveal the behaviour of events during asynchronous communication. In Figure 3.2 (a) events e_0 and e_1 have the same start and finish (date-time) intervals, which indicates that the two events are executed simultaneously. In Figure 3.2 (b) the (date-time) intervals are different which indicates that e_0 executes earlier than e_1.

Figure 3.3 illustrates *synchronous* communication characterizes events e_1 and e_3, indicating their simultaneous execution. Notably, there is inconsistency in the time information for events e_1 and e_3 as the specified start and finish (date-time) intervals differ.

3.3.3 Time consistency in BSO-nets

The verification of date and time consistency in (BSO-nets) involves verifying time consistency between O-nets at various levels of abstraction using the behavioural β and *causal* relationships. The assumption made, for the sake of simplicity, is that all abstraction levels have the same (date-time) origin and granularity.

Given a BSO-net, let *causalU* be the binary relation consisting of the *causally* related pairs of events that are specified as follows:

$$causalU = \bigcup \{causal(e) \mid e \in \mathbf{E}\} \tag{3.12}$$

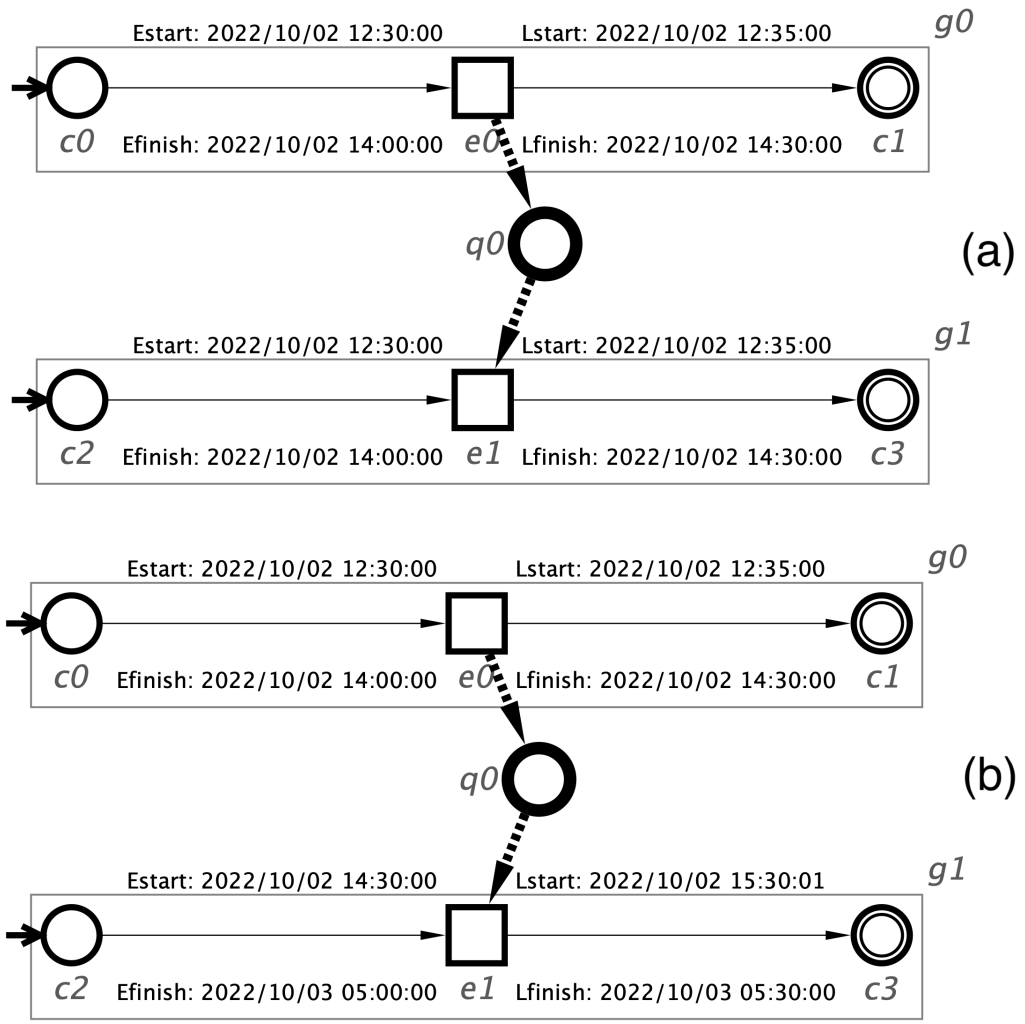


Figure 3.2 Two CSO-nets with different runs of e_0 and e_1.

where **E** is the set of events in the O-nets of the BSO-net. The time information of *causalU* is date and time consistent if and only if the following condition is satisfied:

$$\forall(g,h) \in causalU : (T^g_{s,e} \leq T^h_{s,e} \wedge T^g_{s,l} \leq T^h_{s,l}) \tag{3.13}$$

For all conditions $c_i, c_i' \in \mathbf{C}$ (**C** means the set of conditions in the O-nets of the BSO-net) with $(c_i, c_i') \in \beta$ and c_i belonging to the initial condition of the lower level O-net of the BSO-net, which means that the following equation must be true:

$$I_s^{c_i} = I_s^{c_i'} \tag{3.14}$$

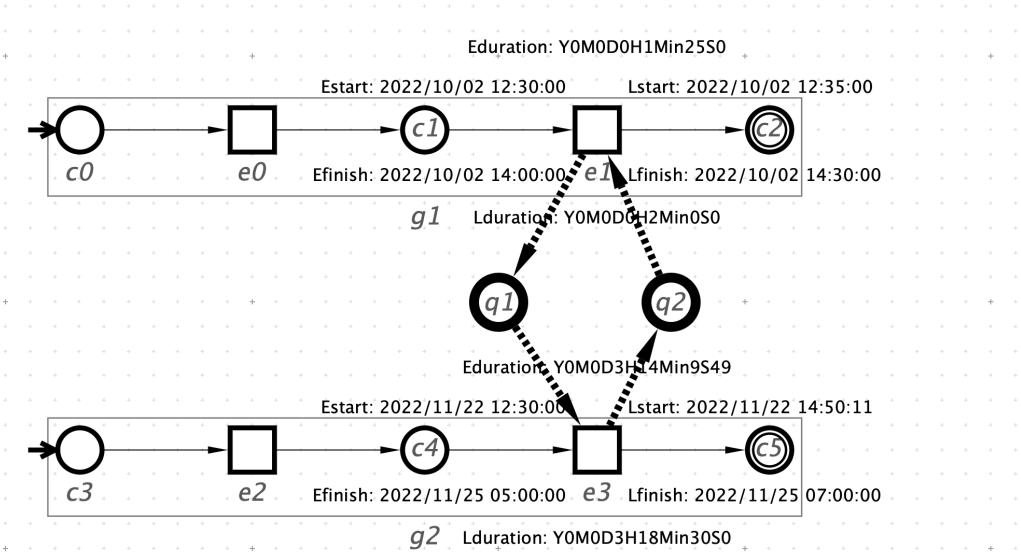


Figure 3.3 CSO-net with time inconsistency

.

In addition, for all conditions c_t, c'_t with $(c_t, c'_t) \in \beta$ and c_t belonging to the final condition of the lower level O-net of the BSO-net, the following equation must hold:

$$I_f^{c_t} = I_f^{c'_t} \tag{3.15}$$

Eq.(3.13) states that there are two events, g and h; the start (date-time) of event g must be the same as, or precede, the start (date-time) of event h. The initial and end states of the BSO-net are constrained by conditions 3.14 and 3.15: the start and finish (date-time) of a lower level O-net must coincide with the start and finish (date-time) of its corresponding upper level condition.

Figure 3.4 shows that the start time interval of post-modified system's (c_4) and the completion time interval of pre-modified system's (c_3) are the same as their respective higher level conditions.

Figure 3.5 illustrates the behaviour of the system update, which reveals inconsistencies in time information. This inconsistency is caused by a difference between the upper-level start time intervals for c_0 and the lower-level start time intervals for c_2.

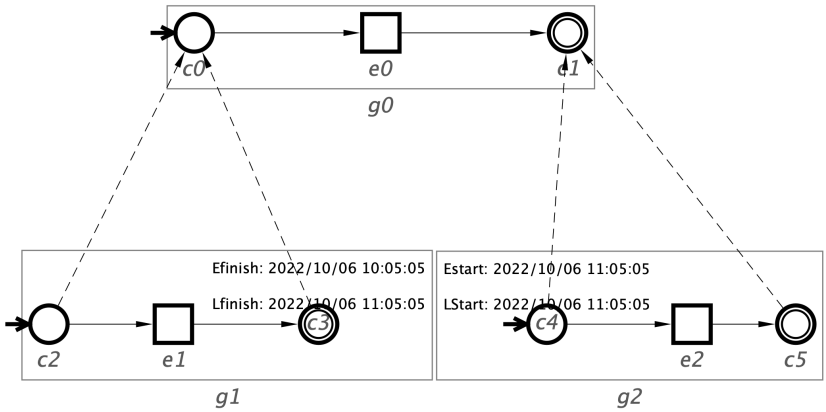


Figure 3.4 BSO-net portraying system update.

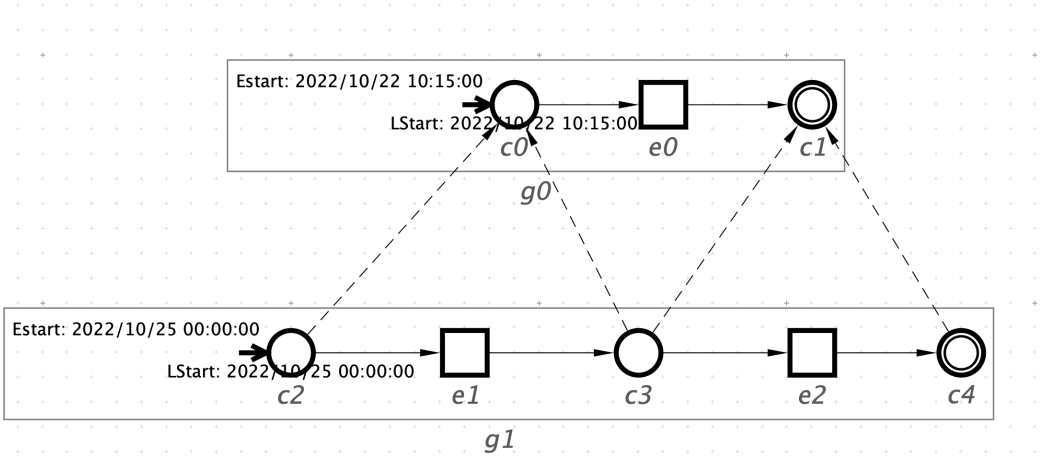


Figure 3.5 BSO-net with time inconsistency.

3.3.4 Time consistency in ALTO-nets

In ALTO-nets, each event has at least one input condition and at least one output condition, and each condition has zero or more input and output events. A condition in ALTO-nets can have multiple input and output events that are in different scenarios. The verification of consistency of the ALTO-nets. Let c be a condition in ALTO-nets the alternative consistency of the temporal information of c is determined by the satisfaction of the following constraints:

$$\exists e \in {}^{\bullet}c(I_f^e = I_s^c) \tag{3.16}$$

$$\exists e' \in c^{\bullet}(I_f^c = I_s^{e'}) \tag{3.17}$$

$$c \textit{ is node consistent} \tag{3.18}$$

Eq.(3.16) states that the start (date-time) interval of c is the same as the finish (date-time) interval of an input event, and the second Eq.(3.17), asserts that the finish (date-time) period of c is equivalent to the start (date-time) interval of an output event. The third Eq.(3.18) asserts that the intervals of start, finish (date-time), and duration of c are satisfied by the equations Eq.(3.6), Eq(3.7), Eq(3.8).

The Figure 3.6 depicts two ALTO-net fragments that possess identical structure and (date-time) but exhibit varying duration intervals. The intervals of c_0 in Figure (a) exhibit alternative consistency with regard to e_0 and e_1, while displaying alternative inconsistent with respect to e_0 and e_2. Hence, it can be concluded that there exists a singular valid scenario in which c_0 is applicable, taking into account the (date-time) information. This is because the execution of e_2 cannot occur after the execution of e_0 due to the presence of time inconsistency. Additionally, in (b), there is no valid scenario for c_1 as it is inconsistent with both its finish (date-time) interval.

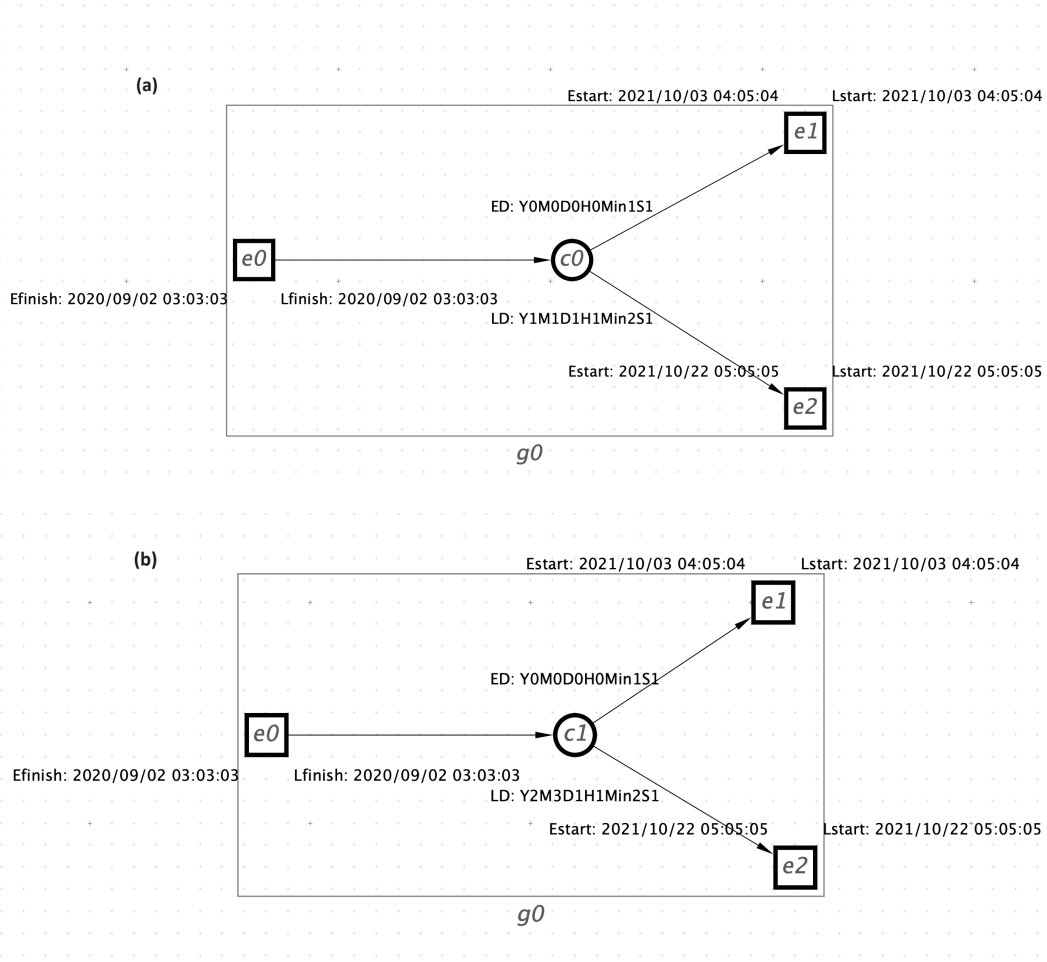


Figure 3.6 Two ALTO-nets with time consistent in (a), and inconsistent in (b).

3.4 Estimation of (date-time) and duration intervals

Investigations of crimes and accidents typically encounter situations where time information is missing, unavailable, or is unknown. In such cases, it is often required to estimate the time information that would have filled the gaps. A missing interval of the node can therefore be estimated by given the specifications of the other two intervals, as illustrated below:

$$[T_{s,e}, T_{s,l}] = [T_{f,e} - D_l, T_{f,l} - D_e] \tag{3.19}$$

$$[T_{f,e}, T_{f,l}] = [T_{s,e} + D_e, T_{s,l} + D_l] \tag{3.20}$$

$$[D_e, D_l] = [max(0, T_{f,e} - T_{s,l}), T_{f,l} - T_{s,e}] \tag{3.21}$$

where $[T_{s,e}, T_{s,l}]$ are the earliest and latest start (date-time) intervals, $[T_{f,e}, T_{f,l}]$ are the earliest and latest finish (date-time) intervals, and $[D_e, D_l]$ are the earliest and latest duration intervals.

When comprehensive time information is available for a node, the precision of the information can be enhanced by employing the following equations:

$$[T_{s,e}, T_{s,l}] = [T_{f,e} - D_l, T_{f,l} - D_e] \cap [T_{s,e}, T_{s,l}] \tag{3.22}$$

$$[T_{f,e}, T_{f,l}] = [T_{s,e} + D_e, T_{s,l} + D_l] \cap [T_{f,e}, T_{f,l}] \tag{3.23}$$

$$[D_e, D_l] = [max(0, T_{f,e} - T_{s,l}), T_{f,l} - T_{s,e}] \cap [D_e, D_l] \tag{3.24}$$

In situations where both date and time intervals of a node are missing, it is necessary to use a specified time interval from another node. The next section describes algorithms that were adapted from [23] for estimating the missing time intervals of the nodes in SO-net using the following two approaches: the first approach is to estimate the intervals of an individual node, and the second approach is to estimate the intervals of all the nodes of the SO-net.

3.4.1 Estimation of finish (date-time) intervals of a node

Algorithms 1, 2 adapted from [23] describe the structure of the procedure *estimateFinish*, which computes the finish time interval of a node n using the *causal* functions and are outlined below.

- In Algorithm 1 given a node n with an unspecified early and late finish (date-time) intervals perform forward breadth first search (BFS) using the $findRightBoundary$ procedure to identify the nodes with a specified early and late finish (date-time) interval that are nearest to n, Figure 3.7.

- In Algorithm 2 using the identified nodes, perform the *backwardBFSDates* (BFS) procedure to calculate the unspecified early and late (date-time), and duration intervals of the nodes causally-related to n, we assume that a default duration interval can be

used as an estimation based on statistics of durations of similar transitions that have occurred in the past (Lines 6-7, 11-14). To increase the node precision, recalculate the intervals (Lines 15-17), until node n is reached.

In Figure 3.7, the early finish (date-time) information and late duration are defined for e_2. The subsequent depiction reveals the calculated results for the unspecified early start (date-time) of nodes causally related to e_2 using Eq.(3.19) .

Algorithm 1 Estimation finish time interval of a node using causal relation

```
1:  procedure estimateFinish(Node n)
2:      RBoundary := ∅ ▷ nearest right nodes of n with specified early, late finish date and
    time intervals
3:      RSector := {n}                          ▷ nodes on paths from n to RBoundary nodes
4:      findRightBoundary(n, RBoundary, RSector)
5:      backwardBFSDates(n, RBoundary, RSector)
6:  procedure findRightBoundary(Node n, Set Boundary, RSector)
7:      Working := {n}                          ▷ nodes used for forward boundary searching
8:      while Working ≠ ∅ do
9:          NextWorking := ∅ ▷ Nodes with unspecified early and late finish time intervals
10:         for all m ∈ Working do
11:             if causalPostset(m) = ∅ then
12:                 add m to Boundary
13:             else
14:                 for all nd ∈ causalPostset(m) do
15:                     add nd to Sector
16:                     if nd.Efinish.specified ∧ nd.Lfinish.specified then
17:                         add nd to Boundary
18:                     else
19:                         add nd to NextWorking
20:             remove m from Working
21:         Working := NextWorking
```

3.5 Estimation of finish time interval for ALTO-nets

In ALTO-net, each condition has zero or more input and output events. A condition in ALTO-net can have multiple input and output events that are different in different scenarios [46]. The Algorithm 3 outlines the structure of the process *estimateFinish* for an ALTO-net. This

Algorithm 2 Estimation finish time interval of a node using causal relation

```
1:  procedure backwardBFSDates(Node n, Set Boundary, SetSector)
2:      Working := Boundary          ▷ nodes used for backward estimation of time intervals
3:      while Working ≠ {n} do
4:          NextWorking := ∅     ▷ nodes with unspecified date, time and duration intervals
5:          for all m ∈ Working do
6:              if ¬m.Eduration.specified ∧ ¬m.Lduration.specified then
7:                  m.Eduration, m.Lduration := defaultDuration
8:              for all nd ∈ causalPreset(m) ∩ Sector do
9:                  add nd to NextWorking
10:                 nd.visits := nd.visits + 1
11:             if ¬nd.Efinish.specified ∧ m.Efinish.specified then
12:                 nd.Efinish := m.Efinish − m.Lduration
13:                 if ¬nd.Lfinish.specified ∧ m.Lfinish.specified then
14:                     nd.Lfinish := m.Lfinish − m.Eduration
15:                 else if m.Efinish.specified ∧ m.Lfinish.specified then
16:                     nd.Efinish := nd.Efinish ∩ (m.Efinish − m.Lduration)
17:                     nd.Lfinish := nd.Lfinish ∩ (m.Lfinish − m.Eduration)  ▷ Eq.(3.22)
18:         for all nd ∈ NextWorking do
19:             if nd.visits = causalPostset(nd) then
20:                 for all ndout ∈ causalPostset(nd) do
21:                     if ¬ndout.Estart.specified ∧ ndout.Efinish.specified then
22:                         ndout.Estart := nd.Efinish
23:                         ndout.Eduration := ndout.Eduration ∩ ndout.Efinish−
24:                         ndout.Lstart
25:                     if ¬ndout.Lstart.specified ∧ ndout.Lfinish.specified then
26:                         ndout.Lstart := nd.Lfinish
27:                         ndout.Lduration := ndout.Lduration ∩ ndout.Lfinish−
28:                         ndout.Estart                                  ▷ Eq.(3.24)
29:             else
30:                 remove nd from NextWorking
31:             Working := NextWorking
```

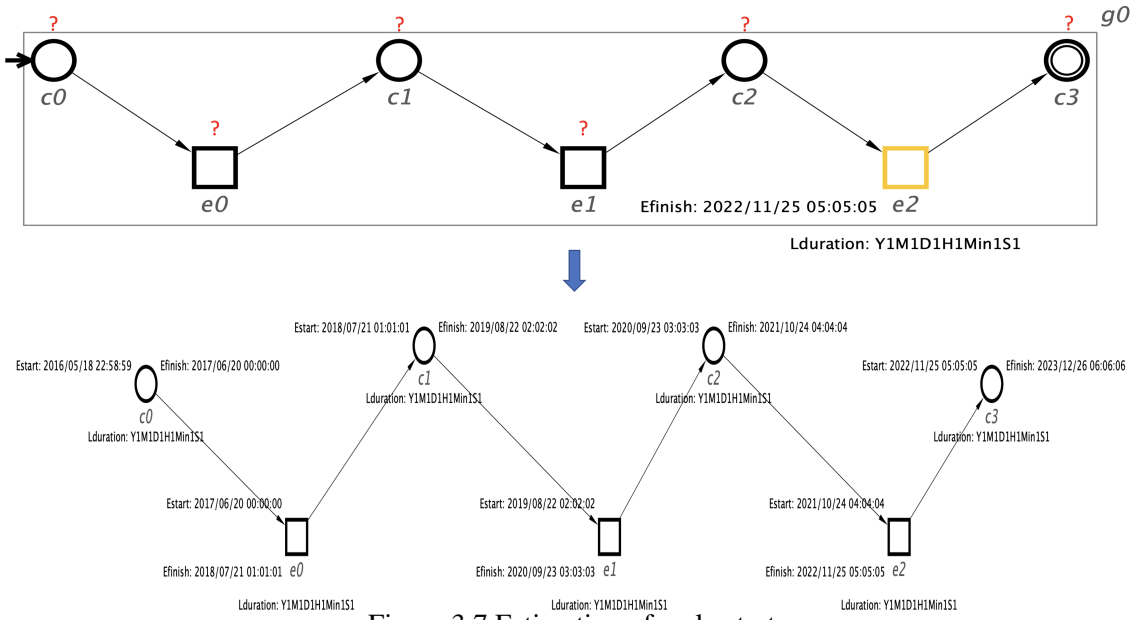


Figure 3.7 Estimation of early start.

procedure calculates the start (date-time) interval of a node n using causal relations. The procedure is described as follows:

- Conduct a forward depth-first search (DFS) for a node n with an undetermined early start (date-time) interval to find all paths that start at n and end to the closest node with a specified early finish (date-time) interval.
- For each path in ALTO-net apply Eq.(3.19), Eq.(3.20), Eq.(3.21), to compute a possible early start (date-time) interval of n, where $I^n_{f,e}$ is the specified early finish (date-time) of the last node in each path and $I^n_{D_l}$ is the late duration of the last node in each path (default late duration interval is used for nodes with unspecified late duration).

Figure 3.8 shows that a forward DFS is used to find all paths from e_0 to the nearest node with a specified early finish (date-time). More specifically, the first path $(c_0, e_0, c_1, e_1, c_2, e_3)$ and the second path $(c_0, e_0, c_1, e_2, c_4, e_4)$, the ALTO-net contains two specified early finish (date-time) interval:

- The early finish (date-time) interval of the node e_3 in the first scenario is: $I^{e_3}_{f,e}$ = [2022/11/04 05:05:05] and the late duration interval of the node e_3 in the first scenario is $I^{e_3}_{D_l}$ = *[Lduration: Y1M1D1H1Min1S1]*, which is subtracted from the specified early finish (date-time) interval of e_3 using Eq.(3.19), Eq.(3.20), and Eq.(3.21).

- Another possible early finish (date-time) interval is calculated using the second scenario. The early finish (date-time) interval of the node e_4 in the second scenario is: $I^{e_4}_{f,e}$ = [2022/11/23 06:06:06] and the late duration interval of the node e_4 in the second scenario is $I^{e_4}_{D_l}$ = *[Lduration: Y1M1D1H1Min1S1]*. Figure 3.8 shows the table for possible times estimations of the early finish (date-time) interval for e_0 using the first and second scenarios. On the other hand, Figure 3.9 shows the table for possible times estimations result for each scenario separately.

Algorithm 3 Estimation finish date interval of a node with alternative using causal relation

1: **procedure** *estimateFinish*(Node n)
2: **Input:***AlternativeON* $(\text{ALTO}-net)$
3: **Output:***AlternativeON* $(\text{ALTO}-net)$ with estimated finish date-time interval of all nodes in each scenario
4: $PossibleTimes := \varnothing$ ▷ possible finish date-time intervals from forward search
5: $visited :=<>$
6: add n to $visited$
7: $forwardDFSDate(visited)$
8: **procedure** FORWARDDFSDATE(*List visited*)
9: **for all** $n \in causalPostset(visited.last)$ **do**
10: $I := null$ ▷ possible finish date intervals
11: **if** $I^n_{f,e}.specified \wedge I^n_{f,l}.specified$ **then**
12: $I := I^n_{f,e} - I^n_{Dl}, I^n_{f,l} - I^n_{De}$ ▷ start date pair of values in Eq.(3.19)
13: add I to $PossibleTimes$
14: **else if** $n \notin visited$ **then**
15: add n to $visited$
16: forwardDFSDates$(visited)$
17: remove $visited.last$

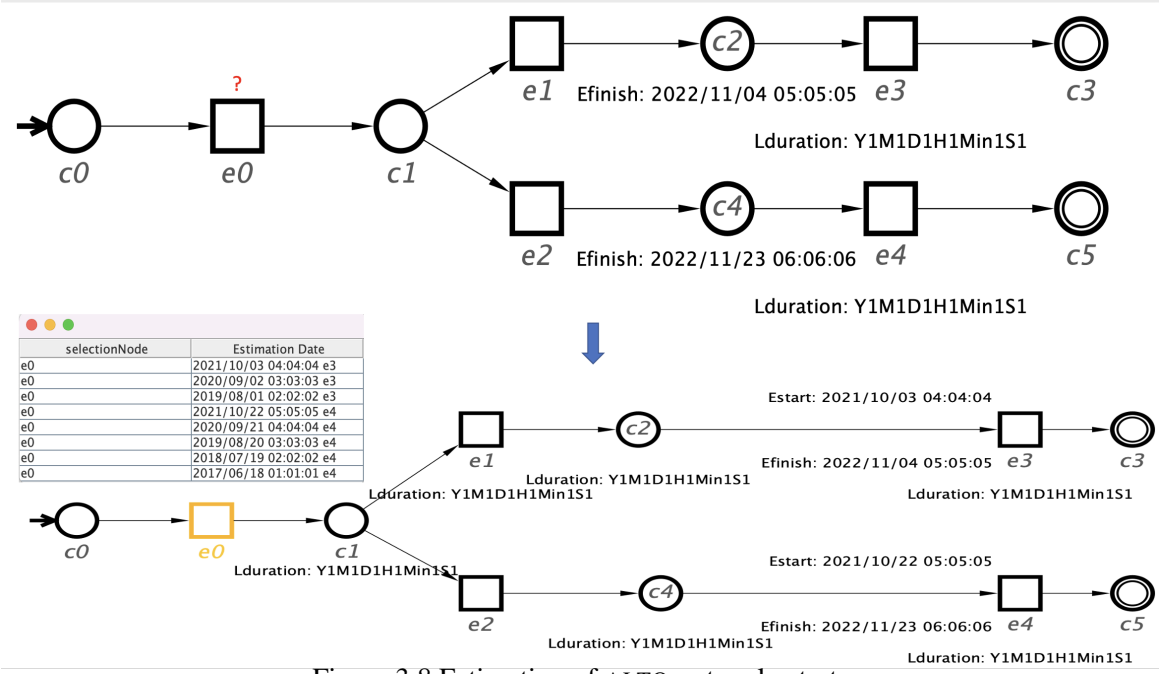


Figure 3.8 Estimation of ALTO-net early start.

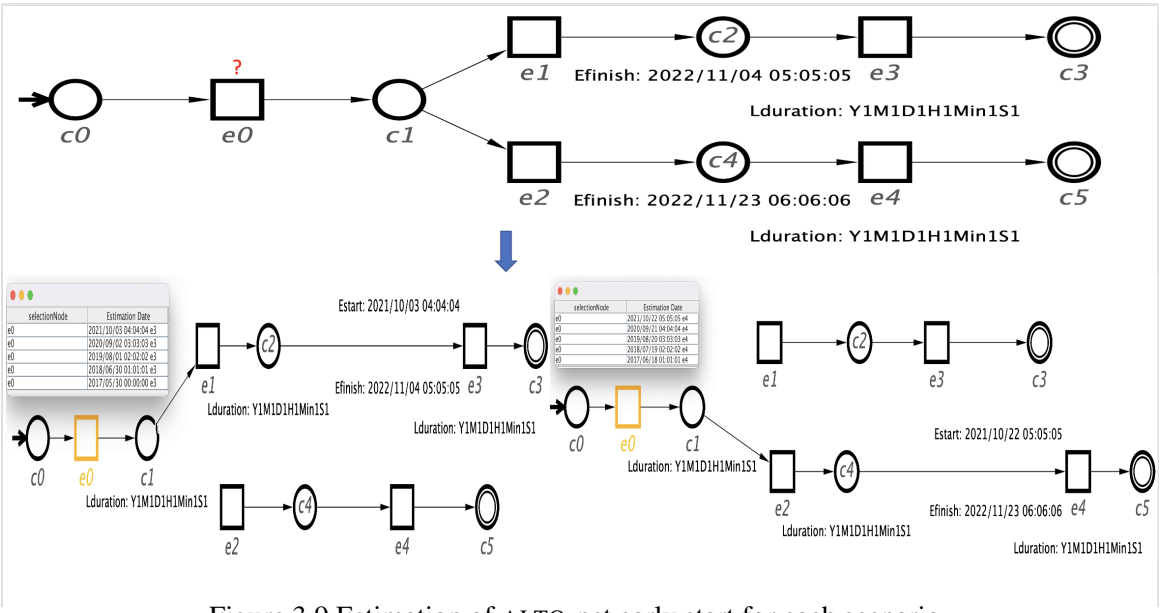


Figure 3.9 Estimation of ALTO-net early start for each scenario.

3.6 Algorithms for checking consistency

This part will examine algorithms used for basic and casual time consistency verification. These algorithms are designed to encode the equations and conditions described in this chapter for the SO-nets and ALTO-net models. To facilitate the readability of the algorithms,

we will denote the start (date-time) with *n.start* of a node n (I_s^n); *n.finish* to denote the finish (date-time) of n (I_f^n); and *n.duration* to denote the duration of n (I_d^n).

The Algorithm 4 is the fundamental function responsible for implementing Equations Eq.(3.6), Eq.(3.7), and Eq.(3.8) in order to validate the consistency of a given node with respect to the specified start, end, and duration intervals. The function's structure is specified by the Algorithm 4. The interval and its estimate is computed on lines 3, 6, and 9, which return FALSE if the intersection is void. As a result, in the absence of a condition being met by the provided time information, the function will produce a FALSE output; in all other cases, it will return TRUE.

Algorithm 4 (Node consistency)

```
1:  function Boolean nodeConsistency (Node n)
2:  I_f^n := n.start + n.duration                          ▷ Eq.(3.20)
3:  if I_f^n < n.finish then                               ▷ Eq.(3.6)
4:      return FALSE
5:  I_s^n := n.finish − n.duration                         ▷ Eq.(3.19)
6:  if I_s^n > n.start then                                ▷ Eq.(3.7)
7:      return FALSE
8:  I_d^n := n.finish − n.start                            ▷ Eq.(3.21)
9:  if I_d^n < n.duration then                             ▷ Eq.(3.8)
10:     return FALSE
11: return TRUE
```

Algorithm 5 first verifies *Concurrent Consistency* with respect to the late finish (date-time) interval of the input places of the given event, then with respect to the early start (date-time) intervals of the output places of the event, then invokes *nodeConsistency* for the basic consistency checking of the event itself.

In Algorithm 6 asynchronous consistency function is invoked only if a SO-net contains a communication relation. The function applies two conditions for asynchronous and synchronous-based checking.

Formally, let q be a *channel place* and let e, e' be the input and output events of q respectively. The date information of q is defined to be *a/synchronously consistent* if and only if the following conditions are satisfied:

$I_{f,l}^e = I_{s,e}^q$ and $I_{s,e}^{e'} = I_{f,l}^q$.

Algorithm 5 Concurrent consistency

```
1: function Boolean concurConsistency (Event e)
2: for c ∈ •e do
3:     if c.Lfinish ≠ e.Estart then
4:         return FALSE
5: for c ∈ e• do
6:     if c.Estart ≠ e.Lfinish then
7:         return FALSE
8: return nodeConsistency(e)
```

Algorithm 6 A/Synchronous consistency

```
1: function:Boolean a/synchronous consistency(Channel place q)
2: if q.input.finish ≠ q.start ∨ q.output.start ≠ q.finish then   ▷ Eq.(3.9), (3.10)
3:     return FALSE
4: return nodeConsistency(q)                                      ▷ Eq.(3.11)
```

In Algorithm 7 the *behavioural consistency* function in line 4 verifies the consistency of all binary relations in *causalU*, lines 7 and 9 verify the restrictions on the initial and final places of the BSO-net. The return value of the function is all the nodes which are behaviourally inconsistency in the BSO-net.

Algorithm 7 behavioural consistency

```
1: function Boolean bhvConsistency (Relation causalU)
2: Result := ∅                                   ▷ behaviourally inconsistent nodes
3: for (e1,e2)∈ causalU do
4:     if e1.start.early > e2.start.early ∨ e1.start.late > e2.start.late then   ▷ Eq.(3.13)
5:         add e1,e2 to Result
6: for c ∈ C do
7:     if •c = ∅ ∧ c.start ≠ β(c).start then                                     ▷ Eq.(3.14)
8:         add c to Result
9:     else if c• = ∅ ∧ c.finish ≠ β(c).finish then                              ▷ Eq.(3.15)
10:        add c to Result
11: return Result
```

Algorithm 8 displays the *alternativeConsistency* function in line 2. It states that the early start (date-time) interval of c is the same as the late finish (date-time) interval of an input event (e), line 4 states that the late finish (date-time) intervals of c the same as the early start (date-time) intervals of an output event which denoted as (e').

Algorithm 8 Alternative consistency

```
1: function:Boolean alternative consistency (Condition c)
2: if e.input.Lfinish ≠ c.Estart then
3:     return FALSE
4: if e.output.Estart ≠ c.Lfinish then
5:     return FALSE
```

3.7 Related work

The existing research on Petri nets, namely in the areas of uncertainty, consistency checking, and time information estimation, is currently constrained. In the study conducted by the authors in [52], Petri nets are expanded by include temporal intervals that are linked to transitions and defining the ranges of firing delays for these transitions; the first indicates the shortest period of time that must pass after all of a transition's input conditions have been met before the transition can fire, and the other time indicates the longest period of time during which the input conditions can be enabled and the transition does not fire. In [78], the interval timed coloured Petri net (ITCPN) model employs a timing mechanism in which transitions determine a delay specified by an interval, and time is connected with tokens. In [36], time Petri nets are a classical formalism that extend Petri nets with transition-related temporal intervals. They gain from real-time system properties like synchronisation, parallelism, etc. being represented simply. This chapter has extended the work in [23] by incorporating dates into our model SO-nets, which is much more practical for supporting, e.g., crime investigations than the previous number-based attempts. Nevertheless, none of the research incorporate models communication abstraction of events, states, or the use of dates in the models.

3.8 Conclusion

To summarise, time is an important and basic aspect of real-life systems. It tells us, for example, how long something lasts, or how quickly it changes. Therefore, representing (date-time) information about system behaviours is important to answer vital behavioural

question and ensure that the resulting framework can provide effective support for incident investigators. the incorporation of temporal information using dates into SO-nets and ALTO-net provide a noteworthy improvement in the modelling and analysis of dynamic systems. We make it possible to depict processes that change over time more accurately by adding temporal features to SO-nets. This temporal dimension offers a potent analytical tool for examining the order and timing of events, providing information about system behaviour. Moreover, in situations requiring adherence to a specific temporal sequence of events, such as concurrent systems, the incorporation of time into SO-nets is of immense value. A more adequate comprehension of system dynamics is enhanced by the capability to model time-dependent processes within the SO-nets framework.

When modelling a system based on time, such as accidents or crimes, it is important to identify the order in which events have happened and to identify the duration of the events. The concept of a timed SO-nets has been introduced in this chapter to represent and analyse causally connected events and concurrent occurrences in developing systems with ambiguous or lacking temporal information. The notation of timed SO-nets (based on discrete time intervals), and algorithms for estimating and increasing the precision of (date-time) intervals using default duration intervals and redundant time information have been provided in this chapter. In addition, we have discussed algorithms to verify consistency for basic SO-nets and ALTO-net. In this chapter, the challenge was to adapt algorithms and notations from [23] for handling dates and ensure that the model accurately captures the subtleties of time-based information, which is crucial for applications such as criminal investigations. Future work could be done to improve the model's ability to analyse spatio-temporal patterns by adding spatial information to nodes that already have time information. Looking at both where events happen and when they happen, could help us make more accurate guesses and learn more about how space and time relate to each other over time for the crime or incident.

Finally, using the time property that was covered in this chapter, we will examine the time simulation to simulate the behaviour of SO-nets and ALTO-net in the following chapter (Chapter 4).

Chapter 4

Time Simulation in SO-Nets

This chapter outlines the timed simulation behaviours of basic SO-nets and ALTO-net. In addition, we introduce the maximal enabled events (*Firing Steps*) of multiple O-nets.

4.1 Introduction

In Chapter 3, a time property framework was introduced for the basic SO-net and ALTO-net concepts. The objective of this chapter is to present the time simulation for SO-net and its abstraction using the timed structured occurrence nets introduced in the previous chapter. A CE-system is composed of a large number of concurrently acting (sub)systems interacting with each other and with the system environment. Therefore, the simulation of timed behaviours of CE-systems occurs in meaningful ways so that, for example, an incident (crime) investigator can use them effectively. In [16], simulation is a crucial problem-solving methodology that plays an integral role in addressing a wide range of real-world situations. Simulation is a method employed to explain and evaluate the dynamics of a given system, enabling the formulation of hypothetical scenarios and facilitating the development of real systems. It can be used to model both existent and conceptual systems. A key problem of handling complexity when large datasets are involved will be addressed by developing hierarchical approach to simulation based on behavioural abstractions. Time simulation is a powerful tool used in many fields to model and analyse the behaviour of complex systems

over time, such as crimes and accidents. In criminal investigations, time simulation may be an effective technique since it enables investigators to piece together the sequence of events that led to a crime. Investigators may better grasp the circumstances leading up to a crime by simulating time, which can be useful in identifying suspects and acquiring evidence.

This chapter explores the theoretical foundations, which will be expounded upon in four main sections. Section 4.2 defines the firing sequence, notation, and algorithm of timed-interval simulation SO-nets, building on the discrete time interval discussed in Chapter 3. The timed simulation behaviours for maximal enabled events (*Firing Steps*) of multiple (O-nets) are detailed in Section 4.3, and Section 4.4 provides an overview of related studies that explore the simulation of timed behaviours. The concluding remarks are in Section 4.5.

4.2 Hierarchical simulation of timed behaviours

Simulation is a crucial problem-solving technique for tackling a variety of real-world problems and can be used to analyse and explain the behaviour of a system. In a real-time simulation, the simulation is run in discrete time with constant steps, also known as fixed step simulation, as time moves forward in an equal duration. In this part, we introduce the idea of firing sequence and behaviour graph to characterise the action of structured occurrence nets.

4.2.1 Time simulation of SO-nets

Every activity in a system has a time duration interval which is different from zero, and we make the added assumption that all activities complete in a finite amount of time. We will assume that every transition takes a bounded, non-zero amount of time to fire. In the semantics of non-timed O-nets, transitions can fire at any time after they are enabled, removing input token and creating output token. A marking in a timed O-net is insufficient information to fully characterise the system's state. Timing details must also be provided for the transition. This is provided as a clock function that indicates the amount of time that has elapsed since the enabling of each enabled transition. When firing durations are included, the O-net semantic is changed. Each transition has a time associated with it. When a transition

becomes enabled, it removes the input token immediately but does not create the output token until the firing duration has finished

4.2.1.1 Time transition firing in SO-nets

The primary characteristic of timed SO-nets is that transitions are enabled must start their firing immediately; and the firing lasts some time within an interval. In timed SO-nets, an enabled transition is fired in three 'conceptual' steps: the first (immediate) step removes tokens from the input places, the second (temporal) step holds the tokens for the duration of the firing time, and the third (immediate) step moves tokens to all of the transition's output places. For example, as shown in Figure 4.1 early duration interval is displayed in e_0 as *ED: Y10M1D1H1Min1S1* which represent $Year, Month, Day, Hour, Minute, Second$ respectively. When e_0 is fired, the token is held for a countdown duration before being subsequently moved to the output place.

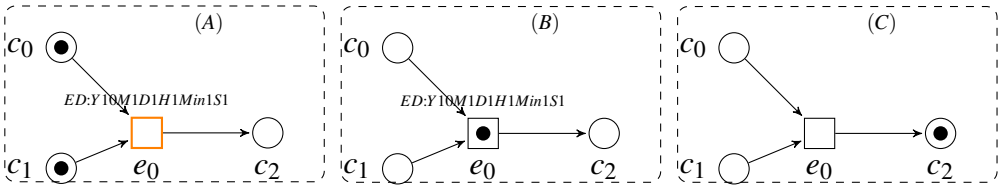


Figure 4.1 Time transition firing in SO-nets.

4.2.1.2 Time transition firing in CSO-nets

This subsection seeks to explain the time simulation of synchronous communication. Figure 4.2 shows a CSO-net that consists of two O-nets interacting with synchronous communication. It shows that in Figure 4.2*(A)*, e_0 and e_1 are enabled and have the same duration intervals, which indicates that the two events are executed simultaneously. Therefore, after firing e_0 or e_1, the transitions will hold the tokens for the duration of the firing time, as shown in Figure 4.2*(B)*, and then moves tokens to all of the transitions' output places, c_2 and c_5, as shown in Figure 4.2*(C)*.

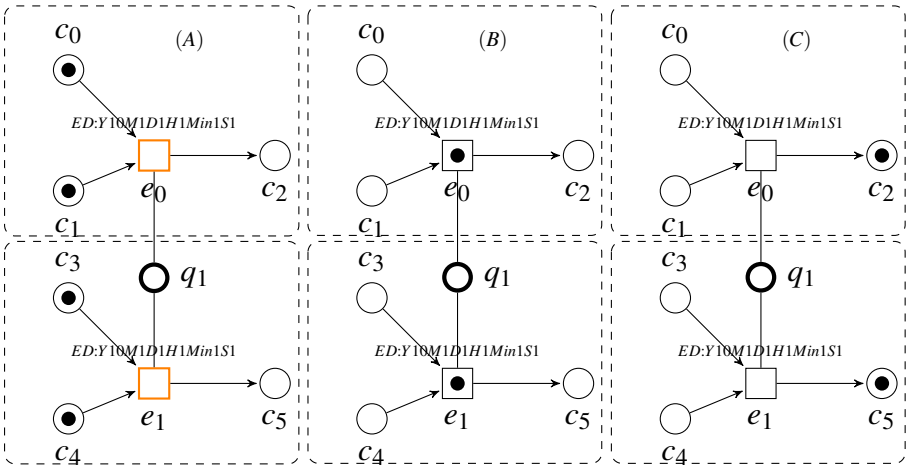


Figure 4.2 Time transition firing in synchronous CSO-nets.

4.2.1.3 Time transition firing in BSO-nets

This section presents the time simulation for BSO-nets, and offers information on the system's development. The representation of the execution history in BSO-nets is divided into two different levels.

Figure 4.3 shows the time simulation firing steps for BSO-nets, and $\{e_0\}\{g_0\}$ is a possible step sequence. As shown in Figure 4.3*(A)*, the only step U_1 enabled at the initial marking $\{a_0, c_0\}$ is $U_1 = \{e_0\}$. In the firing of U_1, the transition holds the token for the duration of the firing time (*ED: Y1M5D1H1Min1S1*), as shown in Figure 4.3*(B)*, and then changes the marking to $\{a_0, c_1\}$, which enables the step $U_2 = \{g_0\}$, as shown in Figure 4.3*(C)*. In the firing of U_2 the transition holds the token for the duration of the firing time (*ED: Y5M5D1H1Min1S1*), as shown in Figure 4.3*(D)*, and produces $\{a_1, c_1\}$ as in Figure 4.3*(E)*.

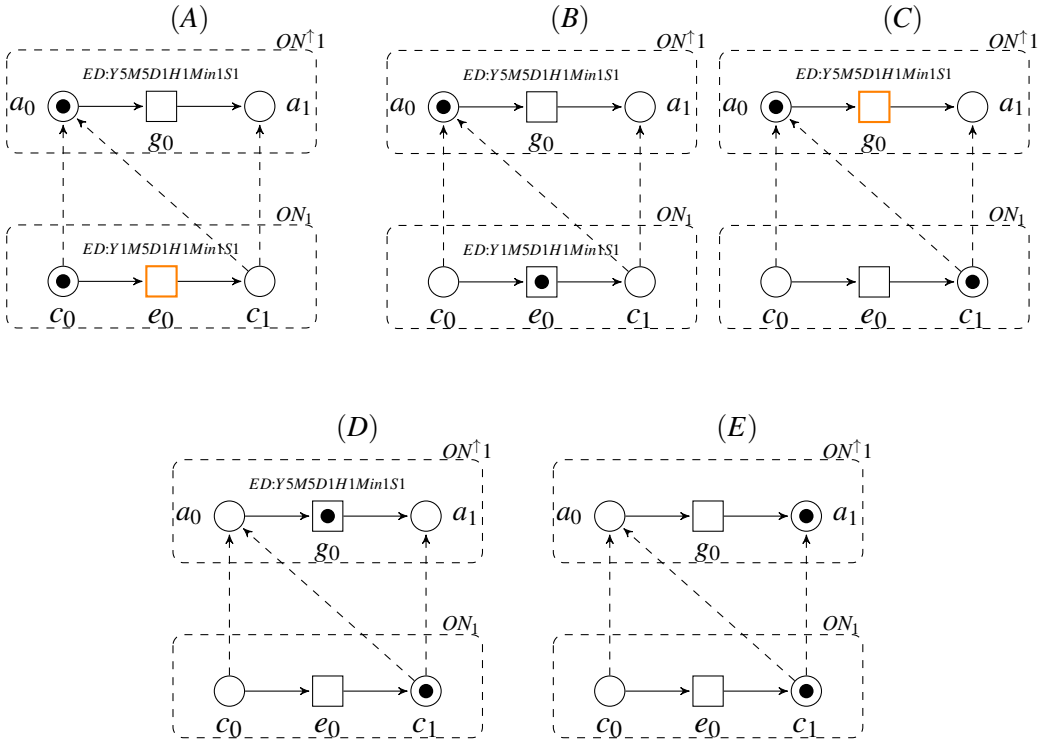


Figure 4.3 Time transition firing in BSO-net.

4.2.1.4 Time transition firing in ALTO-nets

In this section, we introduce the time simulation for ALTO-nets. Each condition contains zero or more input and output transitions, and each transition has at least one input condition and one output condition [46]. Figure 4.4 shows two scenarios for the time simulation for ALTO-nets, first scenario is $(c_0, e_0, c_1, e_1, c_2)$, and the second scenario is $(c_0, e_0, c_1, e_2, c_3)$. The two scenarios are enabled and each one has a different duration interval. As shown in Figure 4.4*(A)*, e_1 and e_2 are enabled and have different duration intervals. When a marked condition generates numerous transitions, just one of these transitions can be selected to fire to simulate the behaviour using time. It should be noted that a step in the ALTO-nets consists of a set of events.

- The shortest duration interval defined for the enabled transitions e_1 is *[ED: Y15M2D1H1Min1S1]*, and e_2 is *[ED: Y20M5D1H1Min1S1]*.

- In Figure 4.4*(B)*, after firing e_1, the token moves and holds in the transition for the duration of the firing time, and then moves tokens to the transition's output place c_2, as shown in Figure 4.4*(C)*.

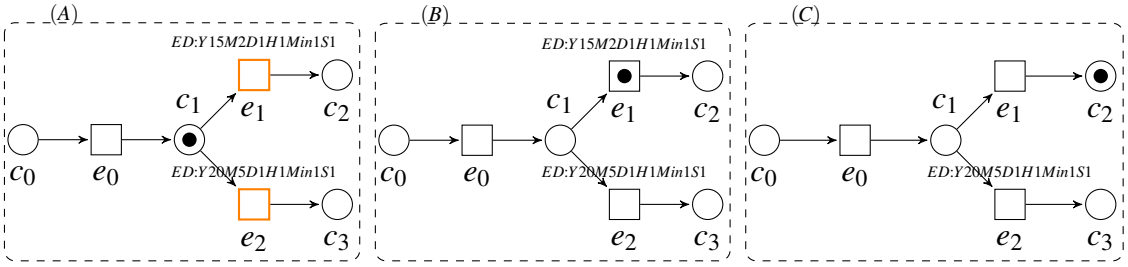


Figure 4.4 Time transition firing in ALTO-net multiple scenarios.

4.2.2 Interval-timed acyclic nets

Timed simulation is a modelling and analytical technique that considers time as a crucial element in simulating dynamic systems. The main feature of interval-timed acyclic nets (described below), which are used for simulation after working out the timings of transitions, is that transitions which are enabled need to start their enabling immediately, and the firing lasts some time within an interval. Thus, the *startfiring* and the *endfiring* of a transition are considered as two distinct events [62].

The firing of an enabled transition in timed simulation is composed of two 'conceptual' steps; the first calculates duration of the firing time, and the second starts to countdown the time units of the duration for the enabled transition [62].

4.2.3 Time semantics

Below $\mathbb{N}$ is the set of non-negative integers, and $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$.

Definition 4.2.1 (interval-timed acyclic net) *An* interval-timed acyclic net *(or* ITA*-net) is a quadruple itanet* $= (P,T,F,I)$ *such that* (P,T,F) *is a well-formed acyclic net, and* I*:*$T \to \{[n,m] \mid n,m \in \mathbb{N}_+ \wedge n \leq m\}$ *is its* interval function.

For every $t \in T$, we denote $I(t) = [\text{sfd}(t), \text{lfd}(t)]$, calling $\text{sfd}(t)$ and $\text{lfd}(t)$ the *shortest firing duration* and the *longest firing duration* of t, respectively.

Definition 4.2.2 (state of ITA-net) *A* state *of an* ITA*-net itanet* $= (P,T,F,I)$ *is a pair* $S = (M,h)$ *such that* $M \subseteq P$ *and* $h \subseteq T \times \mathbb{N}$ *is the* clock relation. *The* initial state *of itanet is* $M^{init}_{itanet} = (M^{init}_{(P,T,F)}, \varnothing)$. *All states of itanet are denoted by* $\text{states}(itanet)$.

The pair $(t,0) \in h$ means that t has just finished its *activity*, and $(t,k) \in h$ with $k > 0$ means that t has still k time units to finish. Moreover, if there is no k such that $(t,k) \in h$, then t is *inactive*. There are three types of events are distinguished : startfire, endfire and tick events. The effect of each of these events on the state of an ITPN is given below.

Definition 4.2.3 (state change rules) *Let* $S = (M,h)$ *be a state of an* ITA*-net itanet* $= (P,T,F,I)$. *The following are possible state-changing events startfire, endfire, tick events and global events:*

- ***Startfire events:*** $S \xrightarrow[+]{U} (M \setminus {}^\bullet U, h \cup \{(t,k_t) \mid t \in U\})$, *where U is a maximal step enabled at M, and* $\text{sfd}(t) \leq k_t(t) \leq \text{lfd}(t)$, *for every* $t \in U$.
- ***Endfire events:*** $S \xrightarrow[-]{V} (M \cup V^\bullet, h \setminus \{(t,0) \mid t \in V\})$, *where* $V = \{t \mid (t,0) \in h\}$.
- ***Tick events:*** $S \xrightarrow{\checkmark} (M, \{(t,k-1) \mid (t,k) \in h\})$.
- ***Global events:*** $S \xrightarrow[\pm]{U:V} S'$, *provided that:* $S \xrightarrow[-]{V} S'' \xrightarrow[+]{U} S''' \xrightarrow{\checkmark} S'$, *for some states* S'' *and* S'''.

A global event $S \xrightarrow[\pm]{U:V} S'$ may be such that $U = V = \varnothing$, in which case only the tick event has an effect. If, in addition, $S = (M,\varnothing)$, then also the tick event has no effect and $S = S'$ and M is a terminal marking of *acnet*.

Definition 4.2.4 (time semantics) *The* time step sequence semantics *of itanet is defined through sequences of global events:*

$$(M^{init}_{itanet} =) S_0 \xrightarrow[\pm]{U_1:V_1} S_1 \xrightarrow[\pm]{U_2:V_2} \dots \xrightarrow[\pm]{U_{n-1}:V_{n-1}} S_n.$$

Algorithm 9 allows to simulate the timed behaviours for O-net. Lines 6 and 7 check if the intervals late start and early finish are specified for the transition node and then calculate the early duration interval for the event. Line 8 displays the duration above the event with colours for each time unit (*Year*: red, *Month*: magenta, *Day*: blue, *Hour*: green, *Minute*: purple, *Second*: cyan). Lines 11 and 12 start firing using the duration interval. The timed simulation steps is specified below. It describes how the transition behaviour change in a system.

1. Check if *latest start (date-time)* and *earliest finish (date-time)* of t are specified.
2. Calculate the *shortest firing duration* then display the time units of (t), *Eduration*: Y, M, D, H, Min, S denoted by *Year*, *Month*, *Day*, *Hour*, *Minute* and *Second* respectively.
3. Remove the token from the *pre-place* to enabled transition (t) to holds the token for the duration of the firing time, then the timer will start by decrementing the time unit.
4. If the time units are finished then the token moves automatically to the transition *post-place*

4.2.4 Case study

In this section, we analyse an example of a timed information and simulation structured occurrence net, using the method proposed above. The evaluation of the proposed solution and validation of its effectiveness is carried out using small size example, the existing medium size case study constructed by MSc student [38].

Scenario 1:

The scenario in Figure 4.5 was interesting because it combines a behavioural abstraction and a synchronous channel place between Oswald and an individual called Donovan. It demonstrates that both Donovan and Oswald are in the US Marines at the same time, they communicate, Oswald gets dishonourably discharged and is then unemployed. Donovan stays in the marines a little longer and then moves on.

Algorithm 9 Timed simulation of O-net

Inputs:
onet - O-net with time intervals
M - current marking
Output:
onet with a step enabled and time units
$t :=$ *enabled transition of onet*
$U :=$ *a step of onet*
for all $t \in \mathbf{T}$ **do**
  if ${}^\bullet t \subseteq M$ **then** ▷ t is enabled
  if $I^t_{l,s}.specified \wedge I^t_{e,f}.specified$ **then**
    $I^t_{e,d} := I^t_{e,f} - I^t_{l,s}$
    display duration in t
    add t to U
for all $t \in U$ **do**
  if $t \in U$ has time units **then**
    $t - firing$ according to its time units
    if t its time units $= 0$ **then**
      $t-$ executed
  $t-$calculate new transition

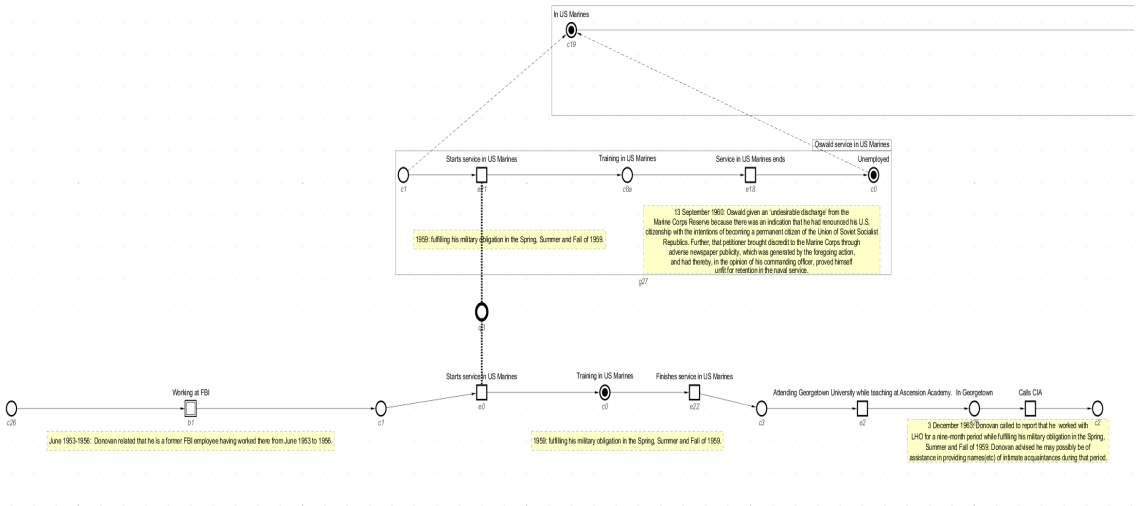


Figure 4.5 Case study

Figure 4.6 illustrates the case study depicted in Figure 4.5, emphasising the temporal aspects. The sequence of firing steps unfolds as follows: initially, events $\{e_1\}$ and $\{e_3\}$ engage in synchronous communication, implying simultaneous firing with identical duration intervals. Subsequently, events $\{e_2\}$ and $\{e_4\}$ become enabled to fire, leading to the subsequent firing of $\{e_5\}$.

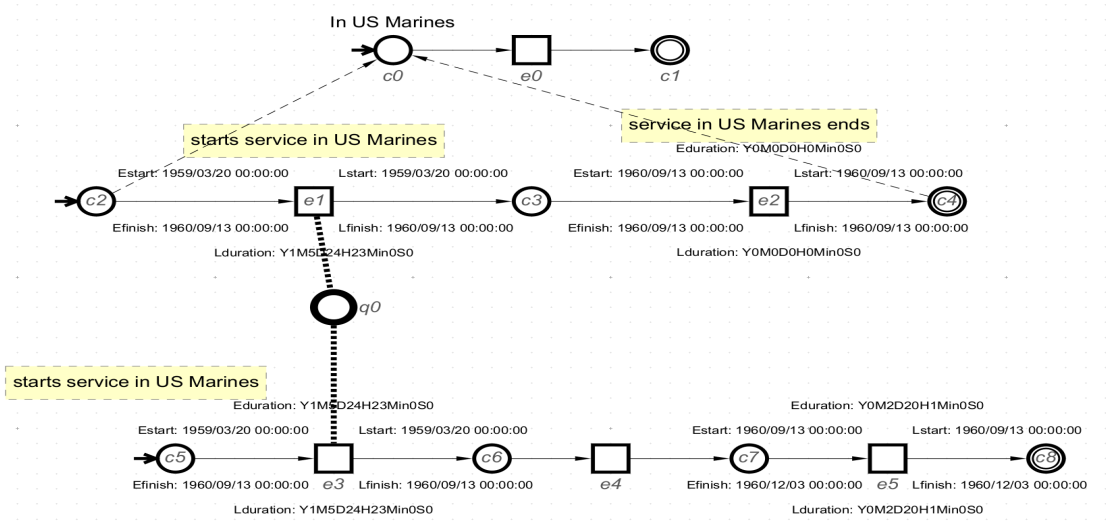


Figure 4.6 Case study with time information

4.3 Maximal events (firing steps)

The term *maximal events* or *firing steps* refers to the sequences sets of transitions that can be executed simultaneously, leading to a state change with no further enabled transitions. The time simulation for a maximal enabled events of multiple O-nets. More precisely the maximal sets of firable transitions are selected so that the enabled transitions in those sets must all fire simultaneously, and the firing of each transition takes a specific amount of time [62]. Understanding *maximal events* is essential for ensuring the correctness, performance, and reliability of concurrent systems represented by SO-nets. This understanding is essential for comprehending how a system evolves over time, especially in scenarios involving concurrent processes.

Algorithm 10 is for implementing *maximal events* (*firing steps*), in lines (6-8) checks if the *late start* and *early finish* intervals are specified in the enabled transitions, then calculates the early duration interval and associates it with each transition. Lines (12-14) start firing simultaneously all the enabled transitions using the *early duration* interval.

In Figure 4.7, we have three different scenarios (A, B and C) with a set of enabled transitions that have different early duration interval for each transition. The early duration interval ED is associated with each transition t, the firing of the transition t takes exactly ED time units ED: Y, M, D, H, Min, S. When a transition starts to fire then its time units start to countdown $I(t)$. If there is a conflict between several enabled maximal sets, the choice is arbitrarily solved. The three different scenarios are generated by different systems and we show how they go through four initial stages of simulation in the same diagram. (We use (st) to indicate the start of firing and (fin) the finish of firing of a transition). In the first step of Figure 4.7 a_{st}, c_{st} , and g_{st} are executed. After that g of the third scenario (C) finishes firing using g_{fin} and h can start firing using h_{st} . Another possibility is to start by firing a_{st}, c_{st} , and e_{st}. After that a of the first scenario (A) finishes firing using a_{fin} and b starts firing using b_{st}.

Figure 4.8 shows a simulation example with synchronous communication in two different scenarios (A and B) with a set of enabled transitions that have different duration intervals for each transition. In synchronous communication, the enabled transitions take the same duration intervals and the two fire simultaneously. Thus, as illustrated in Figure 4.8 , we have

Algorithm 10 Timed simulation maximal sets of enabled transitions

1: **procedure** SIMULATION*maximalSets*
   Inputs:
   ON− set of scenarios (ONs) with time intervals
   M - current marking
2: **Output:**
   ON set of maximal steps (ONs) and time units
   $t :=$ *enabled transition of onet*
3: array set of *maxenabledU* := []
4: **for all** $t \in$ **T do**
5: **if** ${}^\bullet t \subseteq M$ **then** ▷ t is enabled
6: **if** $I^t_{l,s}.specified \wedge I^t_{e,f}.specified$ **then**
7: $I^t_{e,d} := I^t_{e,f} - I^t_{l,s}$
8: display duration in t
9: add t to set of *maxenabledU*
10: **for all** t ∈ set of *maxenabledU* **do**
11: **if** $t \in$ set of *maxenabledU* has time units **then**
12: $t - start\,firing$ according to its time units
13: **if** t its time units $= 0$ **then**
14: $t-$executed
15: $t-$calculate new transition

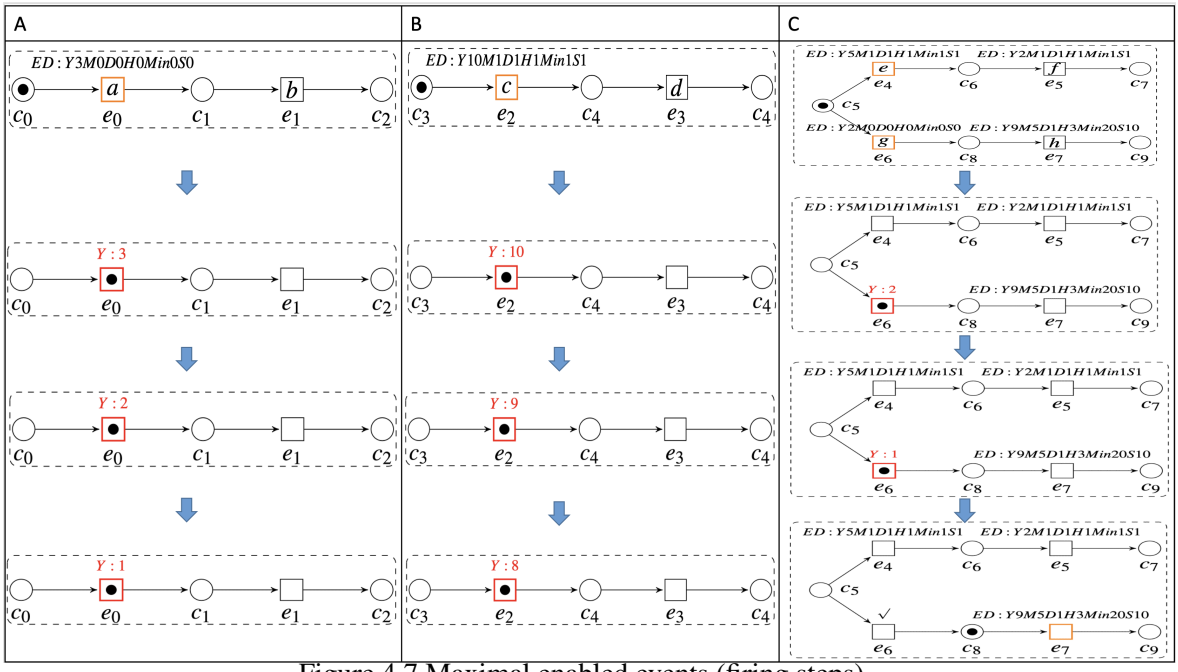


Figure 4.7 Maximal enabled events (firing steps)

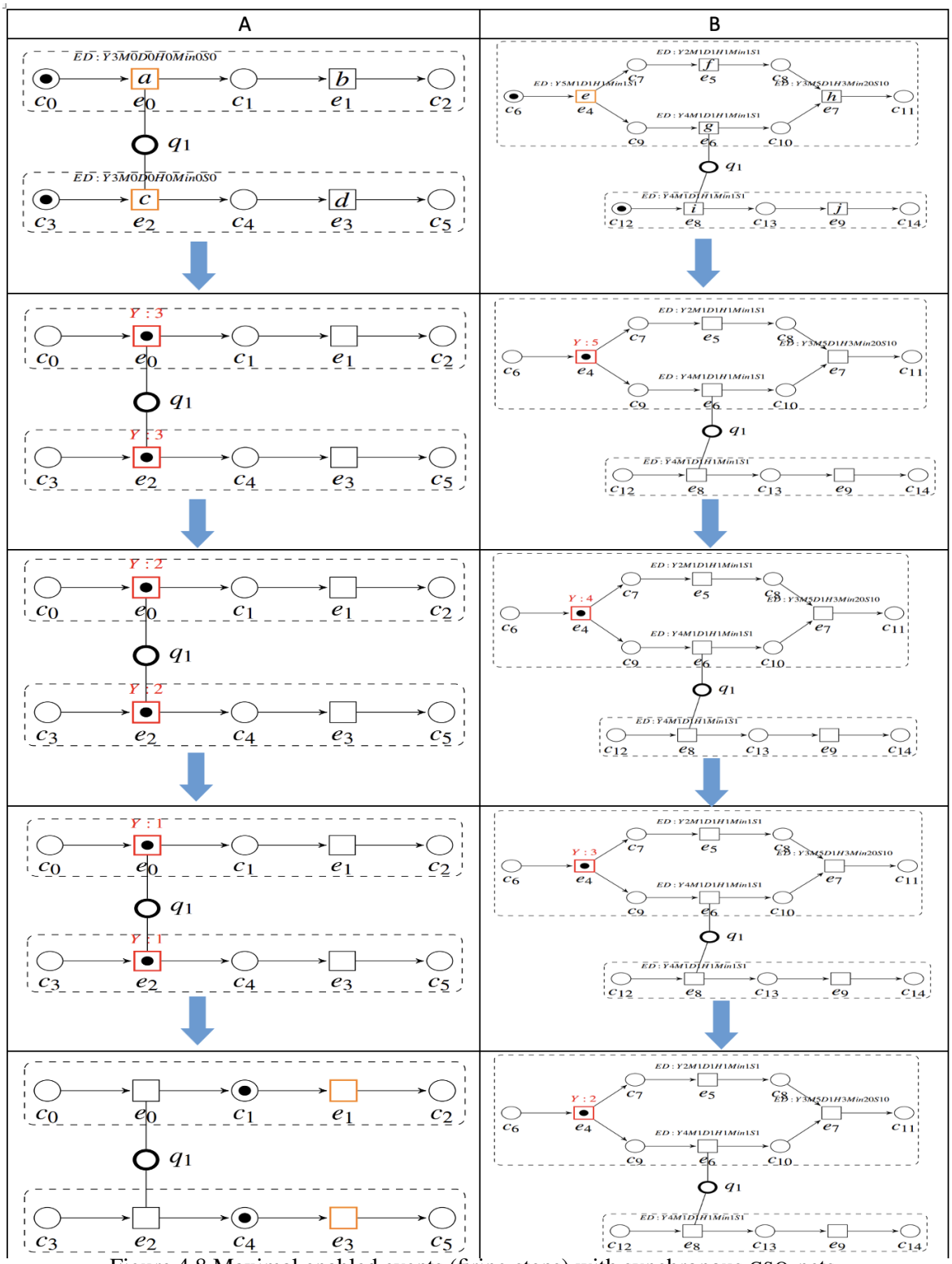


Figure 4.8 Maximal enabled events (firing steps) with synchronous CSO-nets

a_{st} and c_{st} fired simultaneously first, and then a_{fin} and c_{fin} fired simultaneously in scenario (A). After that b and d are enabled to start firing.

4.4 Related Work

We focus in this thesis on the hierarchical simulation of timed behaviours of how a system will behave. In this section, we review several of the related works of time simulation in Petri net models using time, highlighting the key findings and contributions from existing studies, and revealing the current state of the field in which this thesis is situated.

In Timed Petri nets are introduced in [68], the time was associated with transition. For system modelling, where each transition requires a finite, non-zero duration to execute [68] proposed to simulate the speeds of processes or of individual steps in a process. The paper specified a duration d_t to each transition t. If a transition t in a timed Petri net is enabled, it must fire immediately. Thus, the *maximal-step rule* ensures that a maximum number of (just) enabled transitions are always fired. A transition firing cannot be stopped and lasts for d_t time units. The timing of transitions firing events is documented in a tabular format referred to as a firing schedule. A firing schedule for a timed Petri net is a series of initiation and termination periods for net transitions. The firing of a transition is possible if the transition was enabled at the time of the firing. The firing schedule is feasible if every firing in it is feasible. A firing schedule is not feasible if it requires the start or end of an activity earlier than allowed by the end of other activities. A timed Petri net's firing schedule may not be feasible, since some transitions are scheduled to fire even when they are disabled.

In [52], the time attributes were assigned to transitions where each transition has two times specified. The initial value represents the minimum amount of time that must pass since all input conditions of a transition are enabled for that transition to become operational. The other time specified is the maximum amount of time during which the input conditions may be enabled without the transition activating. The transition must fire thereafter. In general, these two periods provide an approximation of the minimum and maximum execution times of the transitions. This paper uses the timed Petri net model to study the recoverability of

communication protocols. It shows that if some a priori knowledge of the events' execution time is not provided, then there are no feasible asynchronous recoverable communication protocols.

In [19], the analysis of concurrent systems is presented in which time appears as a measurable and continuous parameter. Communication protocols are considered as one of the various systems in existence. This study presents an enumerative analysis technique for nets, which involves the computing of a set of state classes and the establishment of a reachability relation on this set. The utilisation of an enumerative technique enables the derivation of a finite representation of the behaviour shown by a broad range of time Petri nets. In the domain of time Petri nets, a common practice involves assigning two distinct time values to each transition. These two values are referred to as the 'static earliest firing time (EFT)' and 'static latest firing time (LFT),' denoting the smallest and largest firing times, respectively, for any given transition. The transition's static firing interval consists of the closed left-bounded interval of times between its static (EFT) and (LFT). As a result, when the net is executed, these intervals are referred to as (dynamic) firing intervals, and their boundaries are referred to as (dynamic) EFTs and LFTs. A temporal Petri net's behaviour is characterised by the set of states it can reach from its beginning state or, conversely, by the set of firing schedules that are possible starting from that initial state.

In the research presented in [26], the preconditions of transition contain more than one token (indicating multiple enabledness of transitions), and then transition remains enabled following its own firing or a firing of a conflicting transition. This scenario is interpreted by assuming that a firing of transition causes a disabling of transition followed by a new enabling, and then the time constraints associated with transition must be applied again after the firing, starting from the firing time. The time Petri net's global state is described in [26] as a pair consisting of a marking as the first component and a set of *clocks* as the second, one for each enabled transition. The clock of a transition starts at zero when it is enabled and increases while time passes. When the clock's value approaches the timing constraint's lower bound, the transition becomes fireable, and is forced to fire when the value of the clock reaches the higher bound of the timing constraint.

The paper [40] introduces timed Petri net simulator software (TiPeNeSS). It can be used to simulate timed Petri nets that include transitions with firing delays that are generally distributed. The functionalities of TiPeNeSS are: investigation of stability in each place and each transition, and steady state simulation. First, transient simulation function is mostly used to check the estimated number of tokens after a determined period of time. The replication/deletion strategy is required for transient simulations to guarantee the simulation's accuracy. Steady state simulation function is employed to determine the steady state behaviour of specific parameters. Finally, stability analysis serves the purpose of ascertaining weather a system parameter has a steady state distribution. Given that stability is a fundamental requirement for conducting steady-state simulation, it becomes imperative to confirm the system's stability. The TiPeNeSS system incorporates a statistical module that ensures the precision of the simulation by enabling the determination of simulation termination based on precision criteria, namely (maximal relative error and confidence level).

The paper [54] presents a discrete-time stochastic Petri net model. The discrete-time SPNs fill the gap between timed Petri nets and normal SPNs. Nonetheless, the introduction of discrete time introduces complexity to the SPN model by allowing the possibility of multiple transitions firing simultaneously within a single time step. This particular type of stochastic Petri net (SPN) is useful in the context of simulating systems that have underlying synchronisation, such as those using a system clock.

The studies conducted by [37, 43, 78] illustrate a high-level Petri net model that incorporates interval timing. A coloured Petri net that is extended with time is referred to as an interval-timed coloured Petri net (ITCPN). Time is represented in tokens, and transitions determine a specific delay for each generated token. The delay is determined by a range of values, consisting of an upper limit and a lower limit, sometimes referred to as an interval. The ITCPN model facilitates the depiction of the dynamic characteristics exhibited by extensive and intricate systems, while also preserving the capability for formal analysis. In the ITCPN model, a timestamp (non-negative integers) is assigned to each token. The timestamp denotes the specific time at which a token becomes available. The natural choice for high-level Petri nets is to associate time with tokens, as this aligns with the existing association of colour

with tokens. The enabling time of a transition refers to the highest timestamp among the tokens that are to be consumed. In addition, the ITCPN model's execution is controlled by the global clock. The model will continue to stay at a specific model time as long as there exist binding components that are both colour-enabled and ready. Several scholarly articles have put forth a Petri net model that incorporates explicit quantitative time (e.g. [2, 82]).

4.4.1 Evaluation

In real-world systems, the execution of activities does not occur instantaneously. Although extensive research has been carried out on time simulation, to the best of our knowledge no single study exists using date-time intervals to simulate the behaviour of the system. In addition, while there are various tools available for simulating Petri nets, we were unable to locate a tool capable of simulating Petri nets using (date-time) intervals. We aimed in this thesis to implement a novel tool-supported formalism (timed SO-nets and timed simulation) for modelling and reasoning about concurrent events with uncertain or missing time information in systems. It is built on collections of connected timed occurrence nets. Timed simulation tools for criminal investigations can help investigators reconstruct the timeline of events related to a crime. The way in which time is represented in our model (using date-time intervals) is much more practical for supporting, for example, crime investigations than previously proposed extensions to deal with time simulation in the literature.

4.5 Conclusion

In conclusion, time simulation has emerged as a powerful tool utilised in several domains to model and analyse the behavior of CE-systems, encompassing applications in the research of crime and accidents. In criminal investigations, in particular, time simulation proves to be effective, empowering investigators to reconstruct the chronological sequence of events that culminated in a crime. By simulating time, investigators gain valuable insights into the circumstances preceding a crime, aiding the identification of suspects and the collection of crucial evidence.

In this chapter, a major contribution was the proposed approach of presenting a timed simulation behaviour for basic SO-nets and ALTO-net. We have introduced the idea of firing sequences and behaviour graphs to characterise the action of SO-nets. The concept of timed simulation behaviour has been introduced to analyse systems, for example, criminal investigations to assist investigators in reassembling the sequence of events and analysing system behaviour. Ensuring that the new execution approach does not disturb the existing notations and definitions developed for the SO-nets was the primary challenge when working with novel execution time semantics to simulate time step-by-step. The coherence and accuracy of the model's original framework has been preserved by carefully integrating the new time semantics.

We defined the firing sequence, notation and algorithm of timed-interval simulation SO-nets. Two 'conceptual' stages comprise the firing of an enabled transition in timed simulation: the first calculates the duration of the firing time, and the second begins to countdown the *time units* of the enabled transition's duration interval. In addition, the second major development was the algorithm and definition of time simulation for maximal enabled events of multiple O-nets. This development is critical to understand how a system changes over time, particularly in situations when multiple processes are running simultaneously.

In the future, more work should be done to improve the time simulation for asynchronous communication in SO-nets. Adding a filter simulation could also be helpful for simulating behaviour using time data, which would allow for a more detailed study of certain events that happened in specific time. It would then be easier to understand how over time, and these improvements could allow the analysis of more complex models.

Checking of time consistency and estimation for timed SO-nets and ALTO-nets, and the development of timed simulation for SO-nets, are discussed in Chapters 3 and 4. The implementation is described in Chapter 5.

Chapter 5

SONCraft: Tool for SO-Nets with Time

This chapter discusses implementations supporting the concept of (date-time) information and time simulation behaviours in the SONCRAFT toolkit.

5.1 Introduction

In Chapters 3 and 4, we described the concept of a time property framework and time simulation in the basic SO-nets and ALTO-net. This chapter provides an account of the practical applications that support these notions. The SONCRAFT toolkit supports the visual editing of timed SO-net models, as well as their verification, temporal simulation, and analysis.

WORKCRAFT [81] is a framework that provides a flexible common underpinning for graph-based models such as Petri nets. SONCRAFT is an open source Java *plug-in* tool that is integrated into the WORKCRAFT platform. It offers various functionalities that facilitate the creation of SO-net based models as well as their animation and verification. An intuitive user interface that promotes efficient use is made possible by a number of implemented algorithms. Also, by employing these tools, users can create SO-net models for various scenarios. Complex behaviours, such as the failure behaviours of CE-systems, can be easily portrayed and analysed. In addition, SONCRAFT provides some initial facilities for entering, editing, validating and simulating three types of SO-nets, namely, CSO-nets, BSO-nets and

ALTO-net. We implemented (date-time) intervals for events and conditions and analysed them in SONCRAFT. We also implemented time simulation using the (date-time) intervals. In this section, we present the implementation of the date-based tools and the time simulation tool.

In this chapter, Section 5.1 presents an overview of the open-source SONCRAFT's plug-in functionality, which helps users to automatically load large amounts of data. Section 5.2 presents the implementation of a time-based SO-net tool and the related analysis to estimate the missing time and check the time consistency. Section 5.3 describes the time simulation tool and the simulation steps. Section 5.4 provides concluding remarks for the chapter.

5.2 Time-based functionality

We implemented a time-based tool for SO-nets and their abstractions, which is described in the following sections.

5.2.1 Time setting tool

The *time setting tool* shown in Figures 5.1 and 5.2 is an interface for specifying date-time and duration intervals of the event and condition nodes in an SO-net model. The early and late duration were specified as follows:

$$(Y{:}Year, M{:}Month,\ D{:}Day,\ H{:}Hour,\ Min{:}Minute,\ S{:}Second).$$

Users can manually set the date information for a selected event in the date panel. For each manually input date, the tool verifies whether or not the date or time is well-defined for each event in the SO-net model. Moreover, for each node, the date is checked.

In addition, the tool provides a calendar for the user to easily set the date of the nodes in the model and the time (*hour*, *minute* and *second*). Moreover, the duration of the node can be set manually or calculated as shown in Figure 5.1.

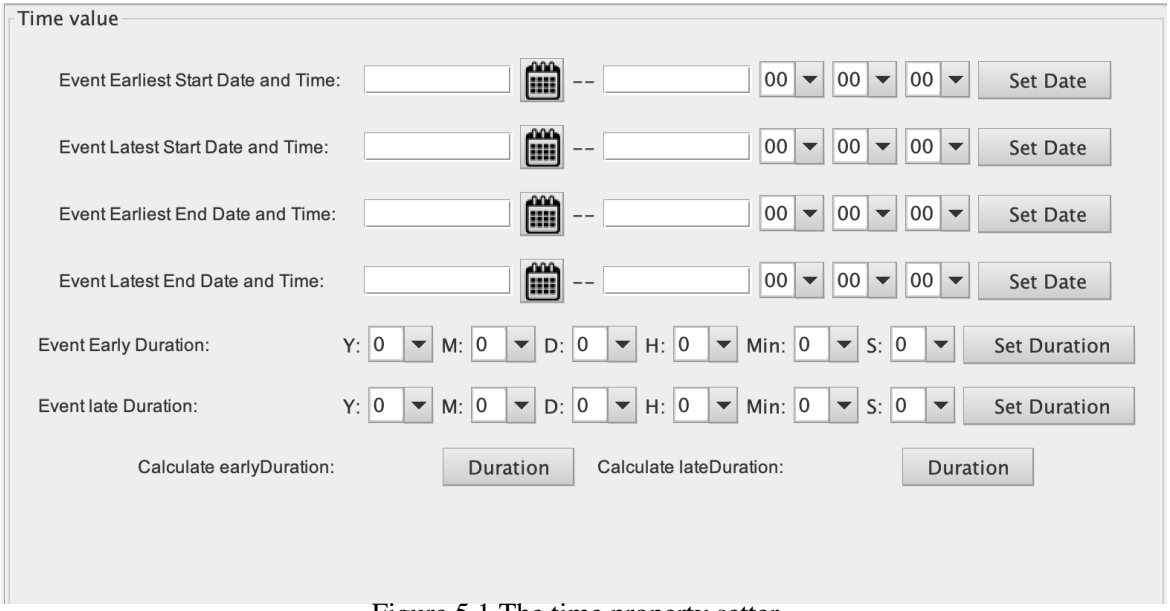


Figure 5.1 The time property setter.

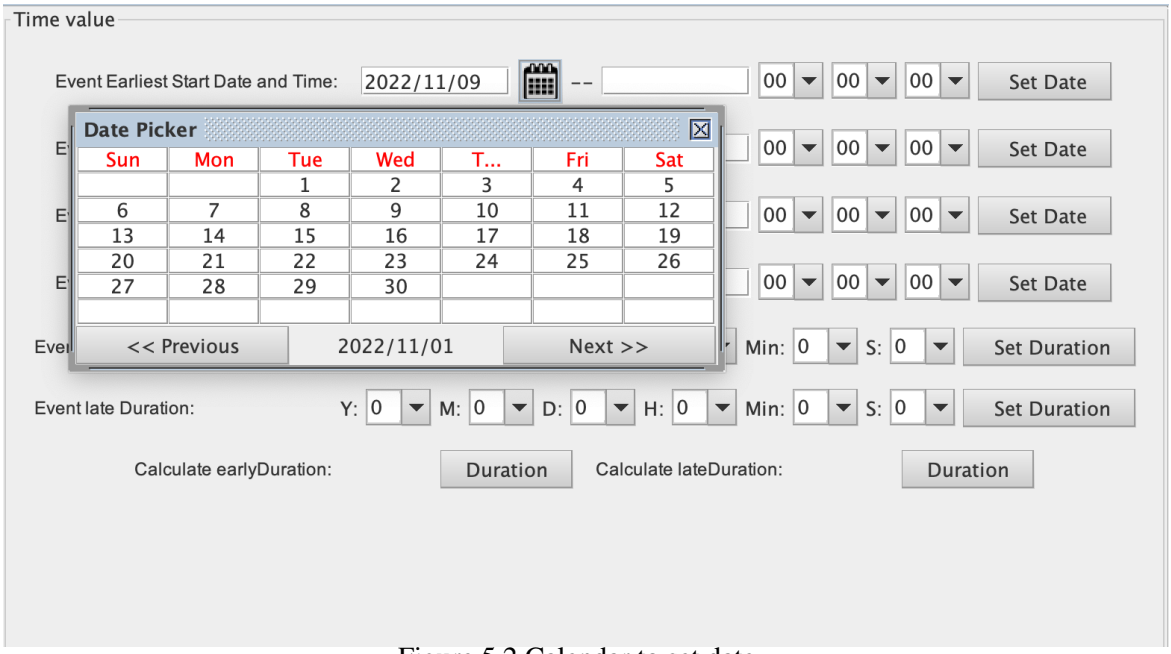


Figure 5.2 Calendar to set date.

5.2.1.1 Time Visualisation

The *time mode* inside the SONCRAFT tool the presentation of temporal information for every event within a SO-net model. Figure 5.3 shows the time representation of an event node displayed with (date-time) intervals (i.e. start, end and duration). Thus, the time information displayed on each event indicates the *Estart*, *Lstart* time and *Efinish*, *Lfinish* time of the event. Thus, each event shows its *Eduration* and *Lduration* value, which the tool calculates from the start and end time intervals of the event.

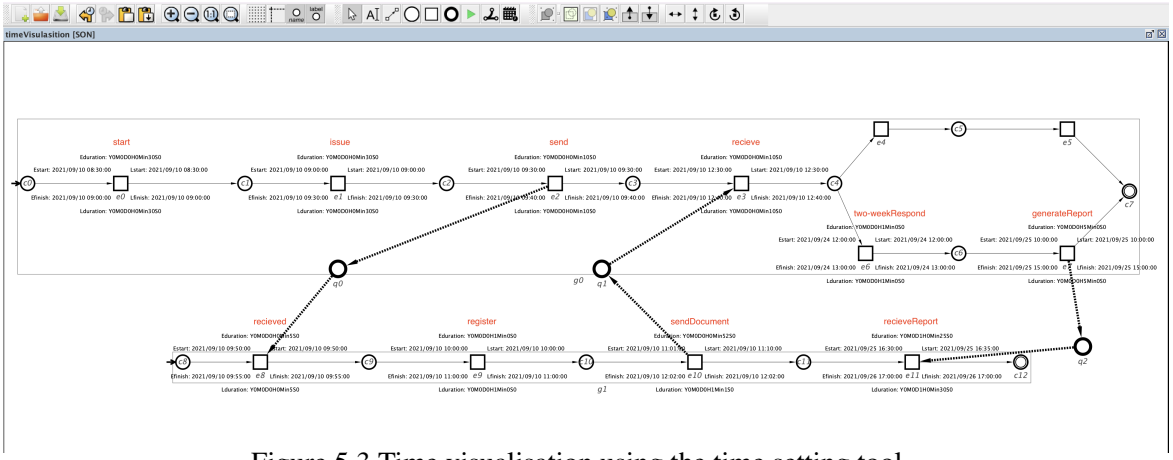


Figure 5.3 Time visualisation using the time setting tool.

5.2.2 Time estimator tool

We implemented a *time estimator tool* designed to handle uncertain time information. In addition, the tool improves the precision of any temporal details provided by the user to the model. Figure 5.4 illustrates an O-net model featuring specified *Efinish* (date-time) information for a chosen node e_1 within the context of the estimator setting window. This window allows users to add an early or late default duration interval for an unspecified duration interval, and then run the estimation for the nodes in the model

5.2.3 Date consistency checking tool

The *date consistency checking* tool provides consistency checking for the time information specified for events in the SO-net model. Figure 5.5 shows the tool performing a consistency

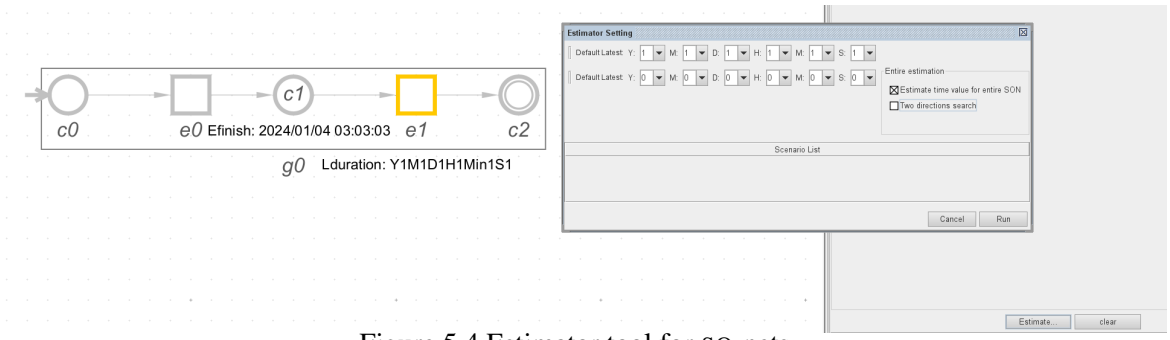


Figure 5.4 Estimator tool for SO-nets.

check for O-net models. The result displays the time information checking task when a node has complete, partial or empty (date-time) information, as in Figure 5.6. Moreover, nodes with a date information and inconsistency errors from any O-net are shown. Figure 5.6 shows one event with a date inconsistency error. For example, the error message involves event e_0 and shows that its start date (*Estart*: 2022/10/04 06:20:02, *Lstart*: 2022/10/04 06:20:02) is later than its end date (*Efinish*: 2022/10/03 00:00:00, *Lfinish*: 2022/10/03 00:00:00); in addition, the consistency verification also checks the consistency for CSO-net and BSO-net models.

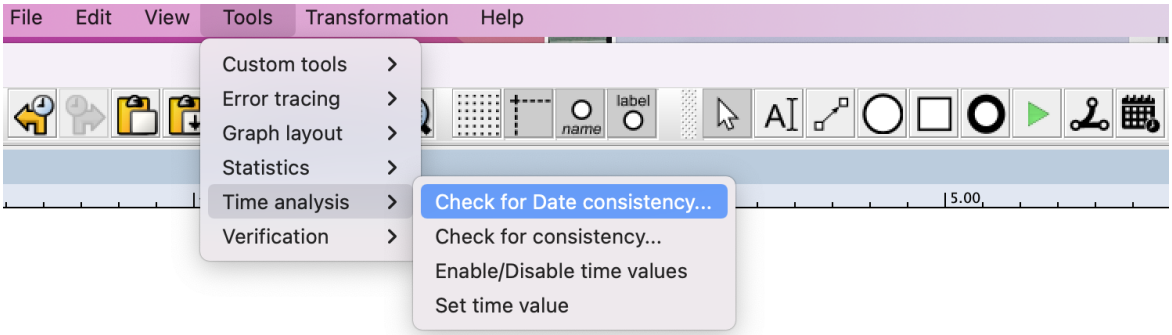


Figure 5.5 Date consistency tool.

Figure 5.7 presents an instance that contains a BSO-net and CSO-net demonstrating inconsistencies in their time information. The depicted scenario highlights the initial error message concerning the behaviour model between O-nets g_0 and g_1. Specifically, it indicates a discrepancy in the (date-time) values of c_0 and c_2, revealing that the commencement date of c_0 (*Estart*: 2022/10/22 10:15:00, *Lstart*: 2022/10/22 10:15:00) does not match the start date of c_2 (*Estart*: 2022/10/25 00:00:00, *Lstart*: 2022/10/25 00:00:00).

Figure 5.7 shows that the second error message involves a *synchronous* communication inconsistency model between g_1 and g_2. It indicates that the (date-time) of e_1 and e_2 and

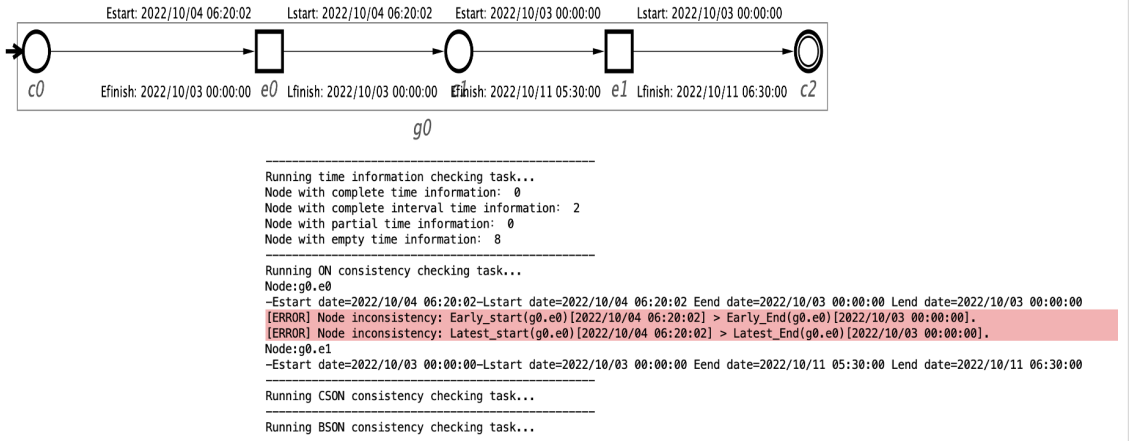


Figure 5.6 Line-like O-net with date consistency checking.

the date of $(g_2.e_2)$ (*Estart*: 2022/11/22 12:30:00) is not equal to $(g_1.e_1)$ (*Estart*: 2022/10/02 12:30:00). Also, the date of $(g_2.e_2)$ (*Lfinish*: 2022/11/25 07:00:00) is not equal to $(g_1.e_1)$ (*Lfinish*: 2022/10/02 14:30:00).

5.3 Time simulation tool

5.3.1 Time simulation in SO-nets

We designed a *time simulation tool* to simulate the behaviour of complex evolving systems using SO-net. Each event in the SO-net has an early and late duration interval, which were used to simulate the time of the event. In the *time simulation tool*, we implemented two buttons to simulate the duration of the event; the early time simulation button was defined as *ETimeSimulation*, and the late time simulation button was defined as *LTimeSimulation*, as shown in Figure 5.8.

The time simulation function in the SO-net *plug-in* can be activated by clicking on either the early or late time simulation buttons in the editor tools panel. If the (date-time) and duration intervals are specified, then the initial marking is automatically set, and all enabled events are highlighted, as shown in Figure 5.9. The time simulation is then conducted manually by clicking the enabled event, after which the calculation for the event duration

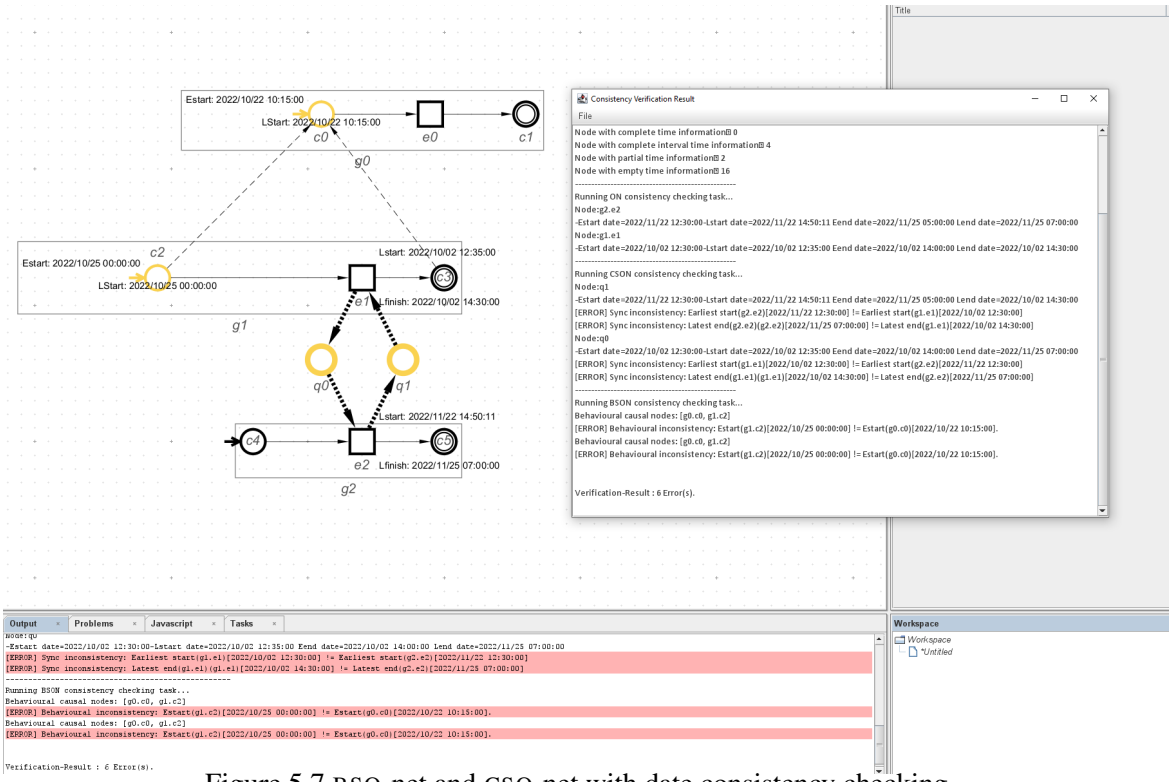


Figure 5.7 BSO-net and CSO-net with date consistency checking.

starts, displaying the result above each event. As shown in Figure 5.10, a countdown of the duration time unit then begins; the durations interval have different colours for each time unit

$$(Year{:}red, Month{:}magenta, Day{:}blue, Hour{:}green, Minute{:}purple, Second{:}cyan).$$

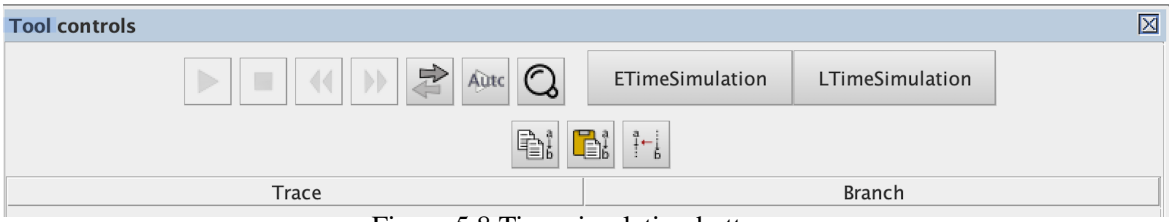


Figure 5.8 Time simulation buttons.

5.3.2 Time simulation in BSO-nets

In this section as shown in Figure 5.11 illustrates another example of time simulation using a BSO-net model. Figure 5.12 shows the time simulation steps. The only step $\{e_1\}$ enabled at the initial marking (c_0, c_2) and the (date-time) intervals are defined as *Lstart*: 2024/02/02

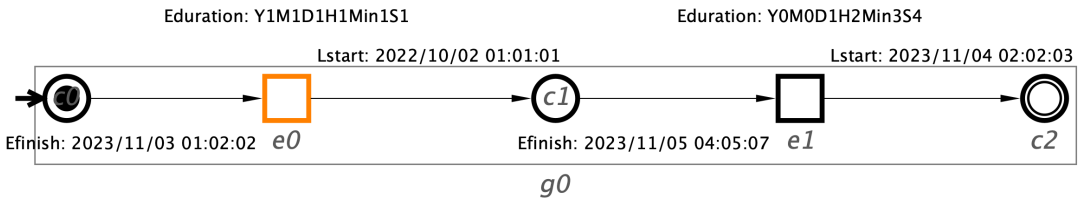


Figure 5.9 Event enabled to simulate the time.

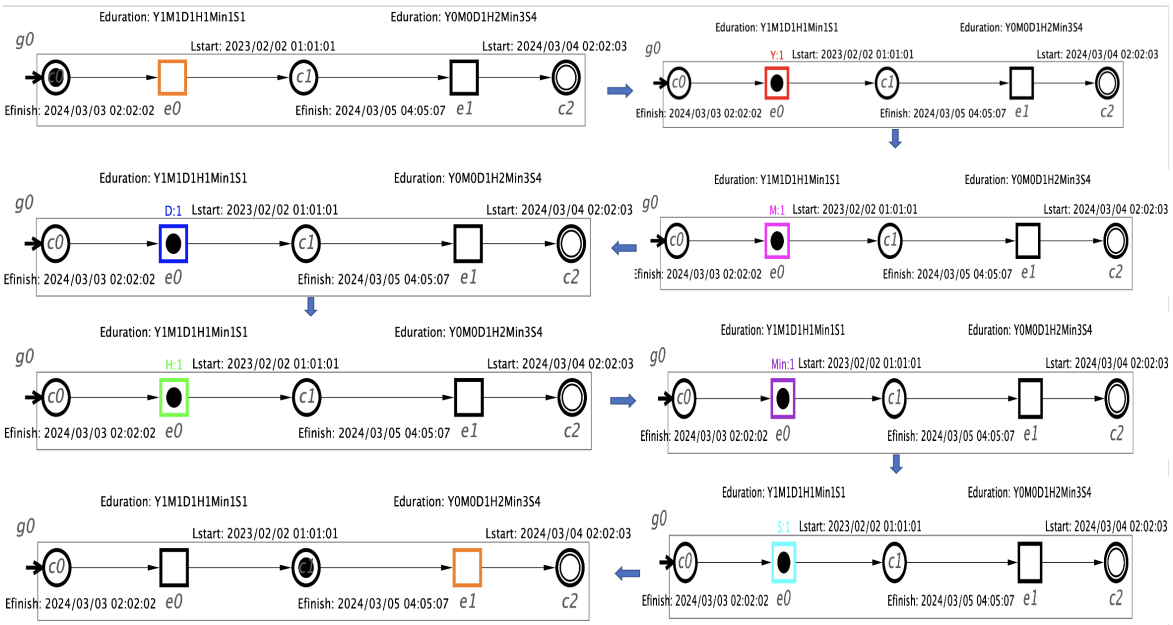


Figure 5.10 *Year*, *Month*, *Day*, *Hour*, *Minute* and *Second*.

00:00:00 and *Efinish*: 2025/03/02 00:00:00. The duration between the late start and early end is early duration *Eduration*: Y1M1D0H0Min0S0, and the firing steps are shown in Figure 5.12a.

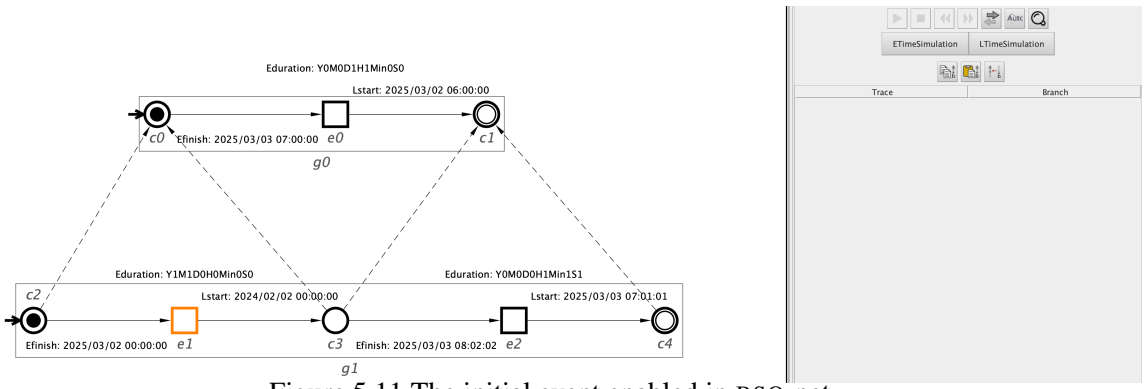
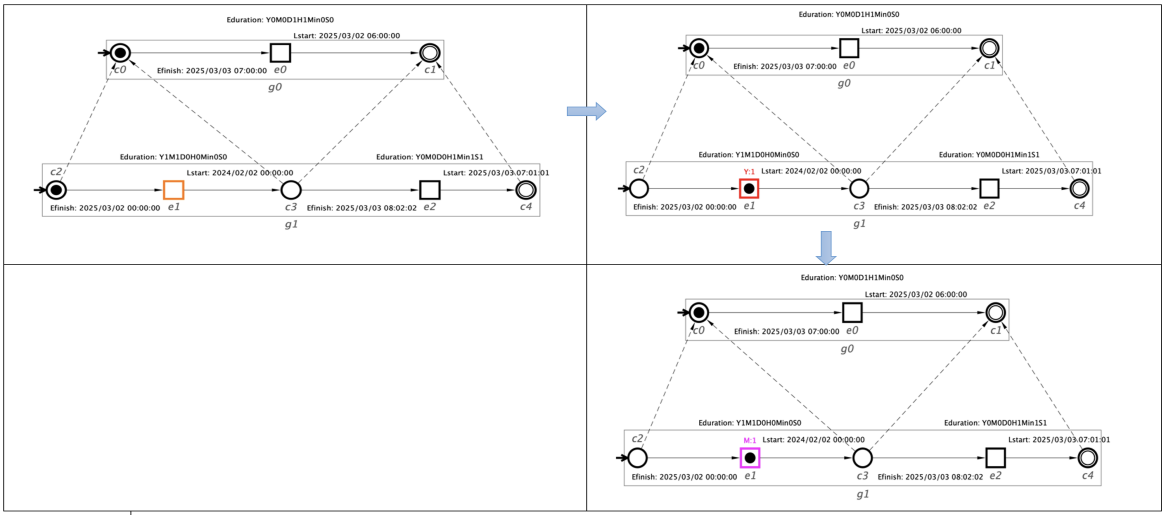


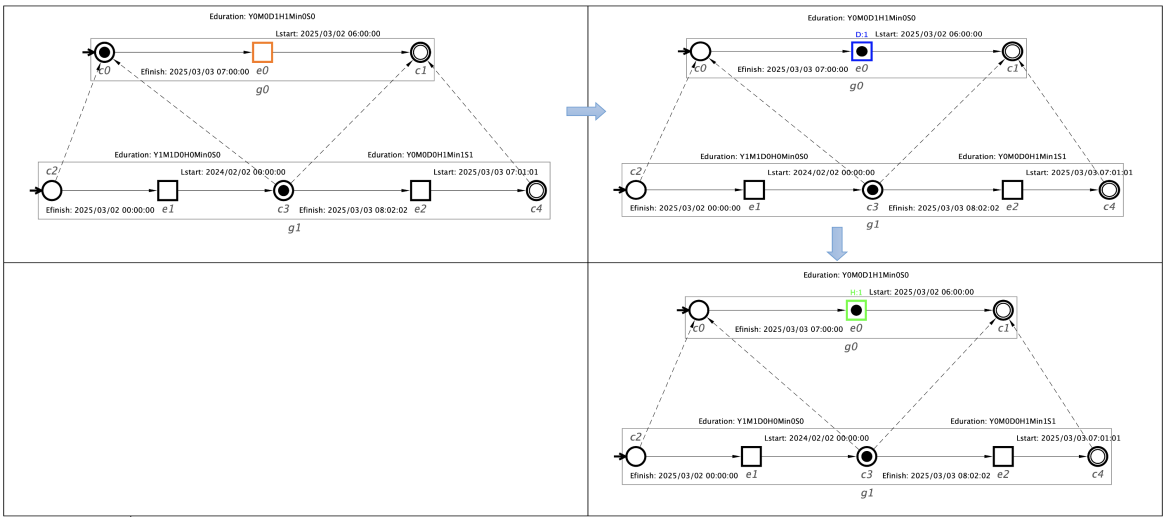
Figure 5.11 The initial event enabled in BSO-net.

The time firing of $\{e_1\}$ changes the marking to (c_0, c_3), which enables $\{e_0\}$. The intervals are delineated as follows: *Lstart* is set at 2025/03/02 06:00:00 and *Efinish* at 2025/03/03 07:00:00. The duration is denoted as *Eduration*: Y0M0D1H1Min0S0, and the firing steps are shown in Figure 5.12b.

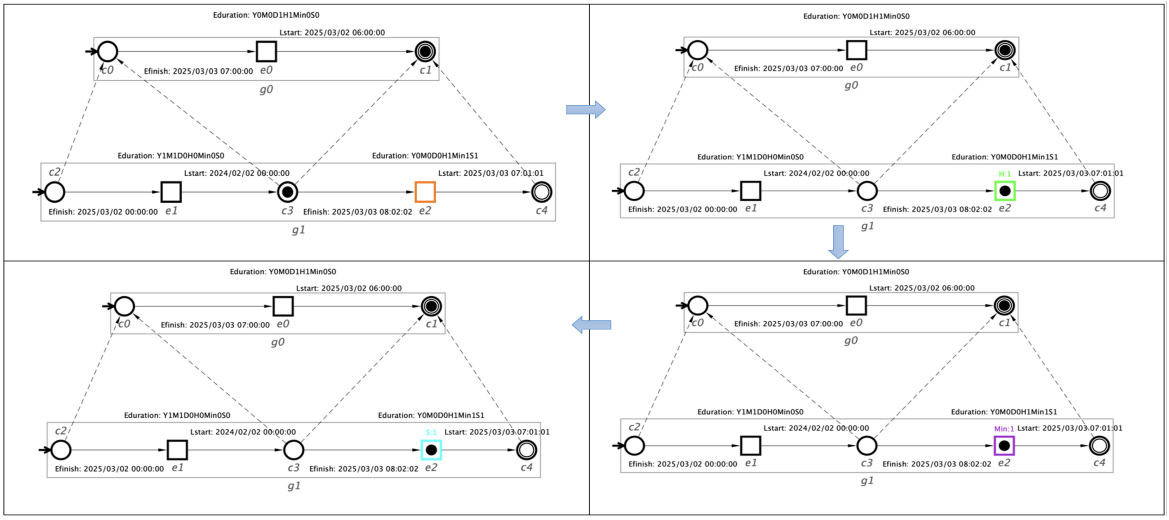
In Figure 5.12c, the time firing of $\{e_0\}$ produces (c_1, c_3) and also enables $\{e_2\}$. The intervals are specified as follows: *Lstart* 2025/03/03 07:01:01 and *Efinish* 2025/03/03 08:02:02. The duration is denoted as *Eduration*: Y0M0D0H1Min1S1, which produces the final marking (c_1, c_4).



(a) e_1 enabled to fire



(b) e_0 enabled to fire



(c) e_2 enabled to fire

Figure 5.12 Time simulation step by step.

5.4 Conclusion

This chapter introduced SONCRAFT, which is a freely available platform utilised for the purpose of modelling, analysing and visualising SO-nets. It is also employed in the process of validating models. The functionalities of SO-nets and their characteristics in SONCRAFT are currently being expanded to encompass new applications, which were introduced in this chapter.

The concept of a timed SO-net tool was implemented in order to represent and analyse causally connected events and concurrent occurrences in developing systems with ambiguous or incomplete temporal information. Moreover, the tool provides efficient date estimation for events with unspecified date information (estimator setting) and a tool to check the consistency of the date information for the entire SO-net, which was introduced in Chapter 3. In addition, we presented a time simulation tool to simulate the behaviour of the model for, e.g., criminal investigations to help investigators reconstruct the timeline of events and analyse system behaviour. The time simulator tool implements the firing steps described in Chapter 4 for simulating models using on SO-nets. Incorporating time information and time simulation into the existing SONCRAFT was the primary obstacle of the work reported in this chapter. In order to ensure that time-based data and time simulations could be accurately represented without disrupting the existing functionalities of the tool, the platform's fundamental structure had to be carefully adapted to incorporate temporal elements.

In the next chapter, we discuss the time granularity for behavioural structured acyclic nets.

Chapter 6

Time Granularity in BSA-Nets

The present chapter discusses the temporal granularity of the two-level behavioural structured acyclic nets (BSA-net). To simulate the behaviour at different levels of time granularity – *minutes* and *seconds* – we consider sequences of phases. Time estimation notation and consistency checks at the two levels are then addressed.

6.1 Introduction

Granularity is crucial when examining real-life systems at various levels of abstraction, as it may be necessary to transition from one level to another to comprehend the observed phenomena [59]. Time granularity, which refers to the degree of accuracy with which temporal intervals are delineated and quantified, is an essential element in a wide range of disciplines, including computer science, physics, finance, and simulation research. Time granularity is the degree of accuracy or precision with which we measure time. The ability to break down time into smaller chunks is crucial, for effective analyses of complex systems. For instance, measuring time in seconds can be most appropriate for analysing low-level (detailed) system behaviour, whereas measuring time in minutes can be most appropriate for analysing high-level (abstracted) system behaviour [55]. Having said that, it is of paramount importance that time information captured using different granularity time scales is kept consistent across the different levels of abstraction. Granularity simulations of timed behaviours is a powerful

tool for analysing complex evolving systems in the case when different time granularity is used at different levels of abstraction.

Time information and simulation were discussed within the SO-net framework in Chapters 3 and 4. These chapters used unstructured time values (positive integers), without considering practical aspects of representing time-related aspects of system such as human comprehension. Chapter 3 proposed representing time using dates as a much more practical way to support crime and incident investigations than the number-based approach. In such cases, it is important to identify the order in which events have happened as well as their duration. Chapter 4 described time simulation tools to help investigators reconstruct the timeline of recorded events.

In this chapter, we discuss theoretical underpinnings, new algorithms, and an implemented prototype software tool for hierarchical and abstraction-based analyses and simulations of timed behaviours of complex evolving systems when different time granularity is used at different levels of abstraction. The discussion is carried out using behavioural structured acyclic nets (BSA-net), which are part of the SO-nets framework. As a result, an incident (crime) investigator could use seconds to analyse the fine details of individual events involved in an incident, and minutes to analyse causal links between these events.

This chapter is subdivided into seven sections. Section 6.2 gives a background about Petri nets and behavioural structured acyclic net. The formalism that is used to support the concept of time granularity in order to model time information in various temporal domains with different levels of detail is defined in Section 6.3. In Section 6.4, we discuss a time granularity model for the two-level BSA-net called TGBSA-net. In Section 6.5, we detail the steps involved in simulating a time granularity behavior of TGBSA-nets. The notation of time estimation for TGBSA-nets and algorithms for estimating and increasing the precision of time in the lower-level granularity using default duration are presented in Section 6.6. Conditions and algorithms for checking the consistency in the presence of different levels of time granularity are given in Section 6.7. A prototype implementation tool support for TGBSA-net with time unit for each level *minute* for the upper level and *second* for the lower

level, and the time simulation behaviour is defined in Section 6.8. An overview of prior literature concerning time granularity is given in Section 6.9.

6.2 Acyclic nets and BSA-nets

In this section, we recall a model based on acyclic nets which generalise occurrence nets. Some of the notations and terminology are the same as in the previous chapters, but one should bear in mind that they can sometimes be formulated differently (or have somewhat different intuitions).

6.2.1 Acyclic nets

Intuitively, acyclic nets are Petri nets that do not use iterative execution. They provide a clear depiction of causality, concurrency and conflict, which are fundamental attributes of concurrent systems [51].

Definition 6.2.1 (acyclic net [4]) *An* acyclic net *is a triple* $acnet = (P,T,F)$*, where* P *and* T *are disjoint finite sets of* places *and* transitions *respectively, and* $F \subseteq (P \times T) \cup (T \times P)$ *is the* flow relation *such that: (i)* P *is nonempty and* F *is acyclic; and for every* $t \in T$*, there are* $p,q \in P$ *such that* pFt *and* tFq*.* ♢

Graphically, places are represented by circles, transitions by boxes, and arcs between the nodes (i.e., places and transitions) represent the flow relation. If it is important to indicate explicitly *acnet*, we denote P, T, F by P_{acnet}, T_{acnet}, F_{acnet}, respectively.

Similarly as for O-nets, to indicate relationships between different nodes, for all $x \in P \cup T$, we denote the *directly preceding* and *directly following* nodes, as follows:

$$^\bullet x = \{z \mid zFx\} \text{ and } x^\bullet = \{z \mid xFz\}.$$

The above notations extend in the usual way to sets of nodes.

The *markings* of an acyclic net *acnet* are defined as sets of places. Moreover, the default *initial* and *final* markings are respectively given by:

$$P^{init}_{acnet} = \{p \in P \mid {}^\bullet p = \varnothing\} \text{ and } P^{fin}_{acnet} = \{p \in P \mid p^\bullet = \varnothing\}.$$

Markings are indicated by drawing black tokens inside the corresponding places.

The execution of acyclic nets follows the standard definition Petri net step semantics. A set of transitions U (a *step*) is *enabled* at a marking M if ${}^\bullet t \subseteq M$ and ${}^\bullet t \cap {}^\bullet v = \varnothing$, for all $t, v \in U$ $(t \neq v)$. This is denoted by $M[U\rangle_{acnet} M'$, where $M' = (M \setminus {}^\bullet U) \cup U^\bullet$. Moreover, a marking M' is *reachable* from marking M if there is a sequence of steps $U_1, \ldots, U_n$ $(n \geq 0)$ and a sequence of markings $M_0 (= M), M_1, \ldots, M_n (= M')$ such that $M_{i-1}[U_i\rangle_{acnet} M_i$, for $i = 1, \ldots, n$. We also denote

$$M[U_1 \ldots U_n\rangle_{acnet} M' \text{ and } M'[\rangle_{acnet} M \,,$$

calling $U_1 \ldots U_n$ a *step sequence* from M to M'. Moreover, all markings reachable from M^{init}_{acnet} are simply called *reachable*.

An acyclic net is considered to be *well-formed* if, for any step sequence that begins from M^{init}_{acnet} it is the case that: (i) no transition is executed more than once; and (ii) no place 'receives' a token more than once.

Note that an O-net is an acyclic net such that $|{}^\bullet p| \leq 1$ and $|p^\bullet| \leq 1$, for every place p. Moreover, all O-nets are well-formed.

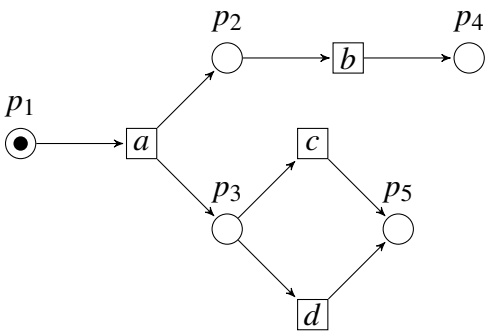


Figure 6.1 Acyclic net

6.2.2 BSA-nets

Behavioural structured acyclic nets (or BSA-nets) can model activities of evolving systems [4]. An execution history is represented on two different levels: the lower-level, which is used to indicate behavioural details, and the upper-level, which is used to represent the stages (phases) of system evolution. Figure 6.2 shows an example of BSA-net [4]. To keep the technical discussion simpler, we do not consider the general BSA-nets. Instead, we consider the case when the upper-level comprises one acyclic net, and the lower-level also comprises one acyclic net.

Definition 6.2.2 (BSA-net [4]) *A* (simple) behavioural structured acyclic net (or BSA-net) *is a triple* $bsan = (lanet, hanet, \beta)$*, where lanet and hanet are well-formed acyclic nets with disjoint nodes and* $\beta \subseteq P_{lanet} \times P_{hanet}$.
Moreover, the following are satisfied (below $\beta_p = \{r \mid r\beta p\}$*, for every place* p *in hanet):*

- $M^{init}_{hanet} = \{p_0\}$ *and* $\beta_{p_0} = M^{init}_{lanet}$.
- *For every* $t \in T_{hanet}$*, there are places* p *and* q *such that* ${}^\bullet t = \{p\}$*,* $t^\bullet = \{q\}$*, and* $\beta_p[\rangle_{lanet}\beta_q$. ♢

Intuitively, *lanet* provides a 'lower-level' view and *hanet* provides a 'upper-level' of alternative records of system behaviour (and if *hanet* is a line-like O-net, then only one such record is represented). The role of β is to identify in the lower-level view the divisions of behaviours into 'phases', and each β_p indicates a 'boundary' between two consecutive phases.

A *marking* of a BSA-net *bsan* is a set of places $M \subseteq P_{lanet} \cup P_{hanet}$, such that $M \cap P_{lanet}$ is a reachable marking of *lanet* and $M \cap P_{hanet}$ is a reachable marking of *hanet*. (The latter means that $M \cap P_{hanet} = \{p\}$, for some place p.) In particular, $M^{init}_{bsan} = M^{init}_{lanet} \cup M^{init}_{hanet}$ is the *initial marking* of *bsan*.

The *phase* of a place p in *hanet* is a set of places in *lanet* given by:

$$phase(p) = \beta_p \cup \{M \mid \exists t \in p^\bullet : t^\bullet = \{q\} \wedge \beta_p[\rangle_{lanet}\beta_q\},$$

and a marking M is *phase-consistent* if $M \cap P_{lanet} \in phase(p)$, where $M \cap P_{hanet} = \{p\}$.

Definition 6.2.3 (BSA-net firing rule [4]) *Let $bsan = (lanet, hanet, \beta)$ be* BSA*-net as in Definition 6.2.2.*

1. *A step in bsan is a set of transitions $U \subseteq T_{lanet} \cup T_{hanet}$. It is* enabled *at a phase-consistent marking M if there is a phase-consistent marking M' such that*

$$\begin{array}{lll} M \cap P_{lanet} & [U \cap T_{lanet}\rangle_{lanet} & M' \cap P_{lanet} \\ M \cap P_{hanet} & [U \cap T_{hanet}\rangle_{hanet} & M' \cap P_{hanet}. \end{array}$$

   We denote this by $M[U\rangle_{bsan} M'$.

2. *A marking M' of bsan is* reachable *from marking M of bsan if there is a sequence of steps $U_1, \ldots, U_n$ ($n \geq 0$) of bsan and a sequence of markings of bsan*

$$M_0(=M), M_1, \ldots, M_n(=M')$$

   such that $M_{i-1}[U_i\rangle_{bsan} M_i$, for $i = 1, \ldots, n$. We also denote

$$M[U_1 \ldots U_n\rangle_{bsan} M' \;\; and \;\; M'[\rangle_{bsan} M \;,$$

   calling $U_1 \ldots U_n$ a step sequence *of bsan from M to M'. Moreover, all markings reachable from M^{init}_{bsan} are simply called* reachable. ♢

Figure 6.2 shows a behavioural structured acyclic net (BSA-net) model of activities of an evolving system [4]. An execution history is represented on two different levels: the lower-level *lanet*, which is used to indicate behavioural details, and the upper-level *hanet*, which is used to represent the stages (phases) of system evolution. The function of β is to delineate the lower-level divisions of behaviours into "phases," and each β_p denotes a "boundary" between two consecutive phases.

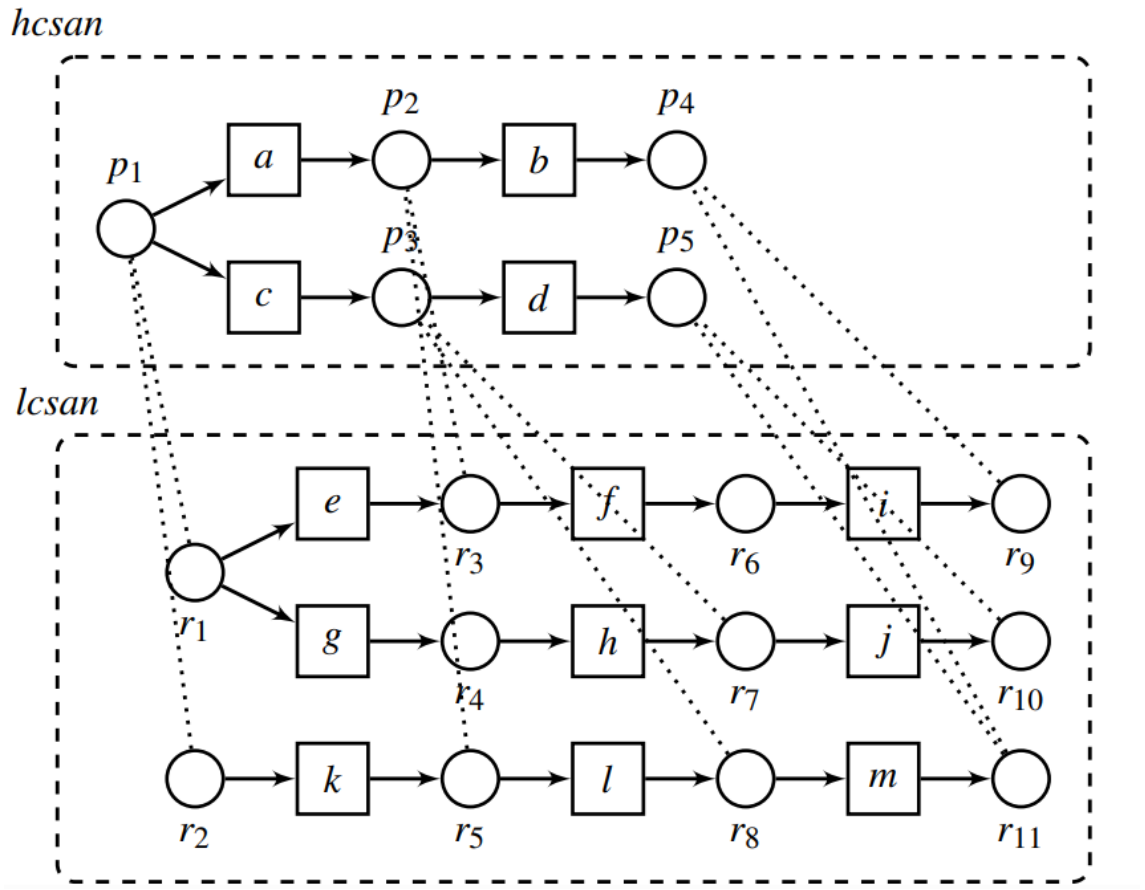


Figure 6.2 Simple BSA-net.

For example, in Figure 6.2, transition c identifies a phase in which transitions g,h,k,l are executed. Moreover, we have:

$$\{r_1,r_2,p_1\}\ [\{g,k\}\rangle_{bsan_0}\ \{r_4,r_5,p_1\}\ [\{h,l,c\}\rangle_{bsan_0}\ \{r_7,r_8,p_3\}\ [\{j,m,d\}\rangle_{bsan_0}\ \{r_{10},r_{11},p_5\}.$$

Hence, $\{g,k\}\{h,l,c\}\{j,m,d\}$ is a step sequence from the initial marking of $bsan_0$ [4].

6.3 Time granularity

The granularity of time is crucial, in particular, for comprehending the operations carried out at coarse levels of time (e.g., abstracted actions) interact with each other [69]. The description of temporal information might vary in terms of abstraction, depending on the desired level of precision and the existing knowledge. The concept of time granularity refers

to the level of detail and precision in the temporal annotation of statements pertaining to executed actions [55]. Moreover, since different levels of abstractions can support different levels of time granularity, it is imperative that the time scales associated with different levels of abstractions are properly related and the behaviours expressed in them are consistent.

6.3.1 Formalising time granularity

Presenting a formal framework that supports the idea of time granularity enables the modelling of time information in relation to temporal domains that have different levels of detail. As in [32], such a formalism can be represented by the *temporal structure*:

$$F = ((\mathcal{T}, \prec, \subset), CONT, \updownarrow) \,,$$

where $\mathcal{T}$ is a temporal universe consisting of a number of disjoint linear temporal layers $\mathcal{T}$ which are totally ordered by the granularity relation $\prec$. Moreover, for all $T \in \mathcal{T}$ and $x, y \in T$, $x \ll y$ means that x strictly precedes y.

For example, we can have $\mathcal{T} = \{Minutes, Seconds\}$ with $Seconds \prec Minutes$. A finer characterisation of the layers is provided by the disjointedness relation $\subset$. As an instance, given $\mathcal{T} = \{Years, Months, Weeks, Days\}$ we have that

$$Months \subset Years \quad Days \subset Months \quad Days \subset Weeks$$

This implies that *Years* are pairwise disjoint when considered as sets of *Months*, and *Months* are pairwise disjoint when viewed as sets of *Days*. However, we do not have $Weeks \subset Months$.

The relation *CONT* links each time point to its belongs layer, and the projection relation $\updownarrow$ establishes a connection between each point and its direct or indirect descendants (downward projection) as well as ancestors (upward projection).

6.3.2 Properties of layers in time granularity

In this section, we recall the properties of layers that are important for describing intricate real-time systems. Understanding the properties of layers in time granularity is vital for enhancing processes, simulations, and analyses within various systems.

Different kinds of temporal structures for time granularity can be characterised through additional conditions on the projection relation. Some of such properties are as follows (below $T, T' \in \mathfrak{T}$): [32]

- **Uniqueness.** The projection relation does not establish a connection between separate points that are part of the same layer:

$$\forall x \neq y \in T : \neg \updownarrow (x, y). \tag{6.1}$$

- **Separation.** If $T' \subset T$ then the decomposition intervals of different points of T are disjoint:

$$\forall x \neq y \in T, x', y' \in T' : (T' \subset T \wedge \updownarrow (x, x') \wedge \updownarrow (y, y')) \implies x' \neq y'. \tag{6.2}$$

- **Order preservation.** The projection relation maintains the linear orderings within layers (in a weak sense, since the projection intervals are ordered, but they might meet):

$$\begin{aligned} &\forall x, y \in T, x', y' \in T' : \\ &\qquad (T' \subset T \wedge \updownarrow (x, x') \wedge \updownarrow (y, y') \wedge x \ll y) \implies (x' \ll y' \vee x' = y'). \end{aligned} \tag{6.3}$$

- **Convexity.** The decomposition relation transforms each point $x \in T$ within a contiguous set into a collection of adjacent points (referred to as the decomposition interval) within the set T'. This criterion eliminates the existence of "temporal gaps" inside the collection of elements of a certain point, as is the case when business months are linked on days.

For any ordered pair of layers T, T' (either $T \prec T'$ or $T' \prec T$), the projection relation links any point of T with an interval of contiguous points of T':

$$\forall x \in T, y, z, w \in T' : (y \ll w \ll z \wedge \updownarrow (x, y) \wedge \updownarrow (x, z)) \implies \updownarrow (x, w). \tag{6.4}$$

- **Total covering.** For any ordered pair of layers T, T', the union of the intervals of T' mapped with the points of T covers T'.
- **Homogeneity.**

  The projection relation establishes that for each pair of distinct layers arranged in a specific level of detail, $T' \subset T$, it links a number of points in the finer layer with each point of the coarser one:

$$\forall x, y \in T : |\{x' \in T' \mid \updownarrow (x, x')\}| = |\{y' \in T' \mid \updownarrow (y, y')\}|. \tag{6.5}$$

6.4 Time granularity model for BSA-nets

The behavioural relation in a BSA-net is a way to capture how the behaviour at a lower-level can be interpreted at a higher (abstract) level. In this section, we introduce a time granularity model for the BSA-net

$$bsan = (lanet, hanet, \beta)$$

defined as in Section 6.2.2, and the time annotations model outlined in Section 3.2 (we assume that these annotations are given in the definition of *bsan*). The idea is to use a suitable temporal structure from Section 6.3 so that time is interpreted in *lanet* and *hanet* using different time scales.

In our discussion, time granularity will be dealt with by the discrete temporal structure

$$F_{min\text{-}sec} = ((\{Minutes, Seconds\}, \prec, \subset), CONT, \updownarrow)$$

where $Seconds \prec Minutes$ and $Seconds \subset Minutes$. Moreover, *CONT* and $\updownarrow$ are defined as usual. Following this, the model of time granularity in *bsan* is introduced as follows:

- Time in *lanet* is interpreted using the $Seconds = \{1s, 2s, 3s, \dots\}$ time domain.
- Time in *hanet* is interpreted using the $Minutes = \{1m, 2m, 3m, \dots\}$ time domain.

Note that $F_{min\text{-}sec}$ is a convex and homogeneous temporal structure.

The resulting *time granularity* BSA-*net* (or TGBSA-net) is then given as:

$$tgbsan = (lanet, hanet, \beta, F_{min\text{-}sec}).$$

In diagrams, time annotations use integers to denote both minutes and seconds (and the calculations involving minutes and seconds are carried out using the corresponding integers). However, their meaning depends on the level in which an annotated node appears: in *lanet*, an integer i denotes the i-th second, whereas in *hanet* an integer i denotes the i-th minute. Thus, for example, 371 in *lanet* (i.e., 371s) corresponds to 7 in *hanet* (i.e., 7m) and so we have $\updownarrow (371s, 7m)$.

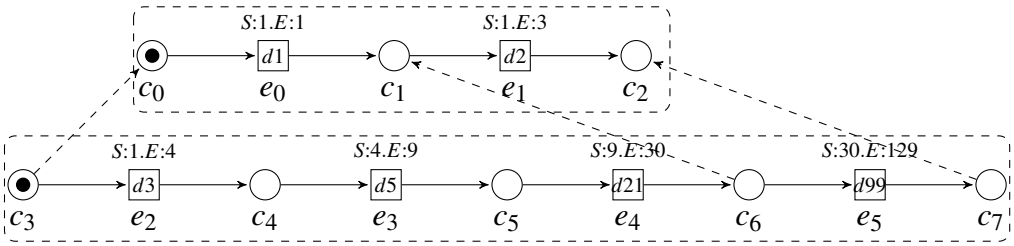


Figure 6.3 TGBSA-net

Figure 6.3 shows a TGBSA-net. The time annotations in the upper-level acyclic net represent minutes. The lower-level acyclic net shows the detailed behaviour of the system using seconds as time units. The dashed arrows between the two levels are used to illustrate the relationships between the lower-level and upper-level behaviours. The start and end times are represented above transitions are denoted by (S and E), and the duration times are represented in the middle for each transition denoted by (d). Note that transition e_0 abstracts three consecutive transitions in the lower-level: e_2, e_3, and e_4. If we simulate the behaviour

of transition e_0 in the upper-level, it takes one minute to execute, whereas the execution of e_2, e_3, and e_4 takes twenty nine seconds in total.

6.5 Time simulation for TGBSA-nets

In this section we discuss time simulation for TGBSA-nets. Time simulation with granularity refers to the representation of time in a simulation using different levels of detail or precision. This concept is crucial in several disciplines, such as computer science, physics, finance, and engineering [71]. In this section we outline the simulation steps for a TGBSA-net $tgbsan = (lanet, hanet, \beta, F_{min\text{-}sec})$. The overall idea is to follow the rules of the operational semantics of the underlying BSA-net $(lanet, hanet, \beta)$ making sure that the timing constraints of *tgbsan* are adhered to, including time granularity.

For example, consider the TGBSA-net in Figure 6.3. One of its possible step sequences which respects time granularity constraints is $\{e_2\}\{e_3\}\{e_4, e_0\}\{e_5, e_1\}$, and Figure 6.4 shows graphically the execution of such a step sequence.

Recall that, in Figure 6.3, the upper-level shows minutes and the lower-level shows seconds. The start and end times are shown above each transition, and the duration is shown in the middle for each transition. Transition e_0 abstracts a set of transitions in the lower-level: e_2, e_3, e_4. When simulating transitions on a second-by-second basis and reaching the last event, the minute-scale transition will automatically move token to the output place. For example, in Figure 6.4, when e_4 is fired, e_0 will execute automatically after it.

Note that the format of displaying dates for the two levels of a TGBSA-net is different as they use different time granularity. More precisely, the lower-level dates use the format:

$$Y{:}Years, M{:}Months, D{:}Days, H{:}Hours, Min{:}Minutes, S{:}Seconds\ ,$$

whereas the upper-level dates use the format:

$$Y{:}Years, M{:}Months, D{:}Days, H{:}Hours, Min{:}Minutes$$

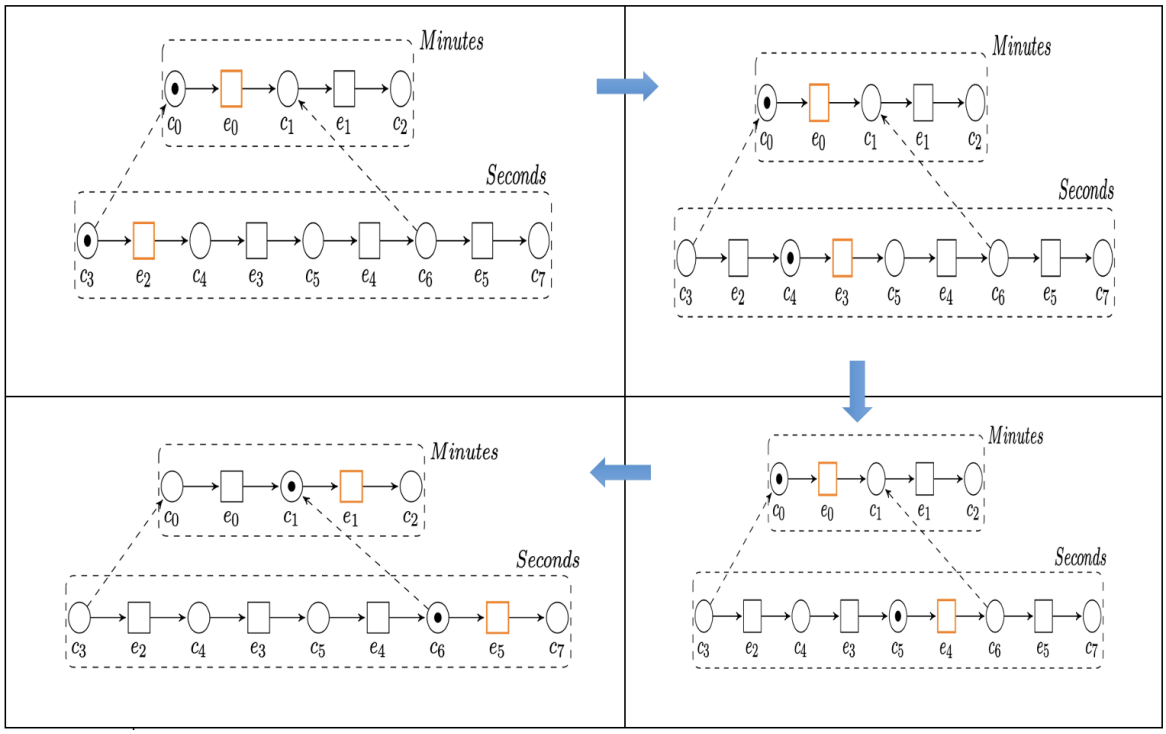


Figure 6.4 Simulation steps for TGBSA-net

Note also that in order to improve the readability of diagrams we can specify, e.g., the seventy fifth second both as [75] and [1:15].

Algorithm 11 describes the time simulation algorithm for behavioural structured acyclic net with time units *Minutes* in the upper-level and *Seconds* in the lower-level. To simulate the time unit we used duration calculated from the start and end time unit in each transition. In lines 13-16 when simulating transitions on a second-by-second basis and reaching the last event, the minutes-scale transition will automatically move token to the output place. Lines 15–16: the transition at the upper level *minute*, which abstracts a set of transitions at the lower level *second*, executes automatically to the output place when the token reaches the final transition at the lower level *second*. Then, the algorithm calculates the next transition to be executed

Algorithm 11 Simulation of TGBSA-net

Inputs:
tgbsan - TGBSA-net
M - current marking
Output:
tgbsan with a step enabled and time units
$U := \varnothing$ a step of *tgbsan*
for all $t \in$ **T do**
  if ${}^\bullet t \subseteq M$ **then**
  if $I^t_{hanet,m}.specified \wedge I^t_{lanet,s}.specified$ **then**
    $I^t_d := I^t_f - I^t_s$
    display duration in t
    add t to U
for all t $\in U$ **do**
  if $t \in U$ has time units **then**
    $t - firing$ according to its time units
    if t its time units $= 0$ **then**
      $t-$executed
      if t_{lanet}.last **then**
        $t_{hanet}-$executed
  $t-$calculate new transition

6.6 Time estimation for TGBSA-nets

Estimating time granularity is crucial in temporal analysis, as it directly impacts the precision and correctness of our comprehension of dynamic systems. As before, we assume that in the upper-level acyclic net time unit are minutes, and in the lower-level acyclic net time units are seconds. The lower-level granularity is used to compute the missing values in time intervals and then mapping the lower-level seconds to the upper-level minutes. Default duration is used for nodes with unspecified duration. For example, as shown in Figure 6.5, where in the lower-level we compute the missing time interval of transitions.

In Figure 6.5(a), the missing time is the start and end of e_1 and e_2, the missing time in e_3 is the end, and in e_4 the missing time is the start. Algorithms 12 and 13 presented below compute the missing time information for Figure 6.5(a) using the causal relation and Eq.(6.6), where the missing interval of a transition e is possible to estimate if the other two intervals are provided, as follows:

Algorithm 12 Estimation finish time interval of a node using causal relation

```
1: procedure estimateFinish(Node n)
2:     RBoundary := ∅ ▷ nearest right nodes of n with specified early, late finish date and time intervals
3:     RSector := {n}                          ▷ nodes on paths from n to RBoundary nodes
4:     findRightBoundary(n, RBoundary, RSector)
5:     backwardBFSDates(n, RBoundary, RSector)
6: procedure findRightBoundary(Node n, Set Boundary, RSector)
7:     Working := {n}                          ▷ nodes used for forward boundary searching
8:     while Working ≠ ∅ do
9:         NextWorking := ∅ ▷ Nodes with unspecified early and late finish time intervals
10:        for all m ∈ Working do
11:            if causalPostset(m) = ∅ then
12:                add m to Boundary
13:            else
14:                for all nd ∈ causalPostset(m) do
15:                    add nd to Sector
16:                    if nd.Efinish.specified ∧ nd.Lfinish.specified then
17:                        add nd to Boundary
18:                    else
19:                        add nd to NextWorking
20:            remove m from Working
21:        Working := NextWorking
```

Algorithm 13 Estimation finish time interval of a node using causal relation

```
1:  procedure backwardBFSDates(Node n, Set Boundary, SetSector)
2:      Working := Boundary          ▷ nodes used for backward estimation of time intervals
3:      while Working ≠ {n} do
4:          NextWorking := ∅     ▷ nodes with unspecified date, time and duration intervals
5:          for all m ∈ Working do
6:              if ¬m.Eduration.specified ∧ ¬m.Lduration.specified then
7:                  m.Eduration, m.Lduration := defaultDuration
8:              for all nd ∈ causalPreset(m) ∩ Sector do
9:                  add nd to NextWorking
10:                 nd.visits := nd.visits + 1
11:             if ¬nd.Efinish.specified ∧ m.Efinish.specified then
12:                 nd.Efinish := m.Efinish − m.Lduration
13:                 if ¬nd.Lfinish.specified ∧ m.Lfinish.specified then
14:                     nd.Lfinish := m.Lfinish − m.Eduration
15:                 else if m.Efinish.specified ∧ m.Lfinish.specified then
16:                     nd.Efinish := nd.Efinish ∩ (m.Efinish − m.Lduration)
17:                     nd.Lfinish := nd.Lfinish ∩ (m.Lfinish − m.Eduration)    ▷ Eq.(6.7)
18:         for all nd ∈ NextWorking do
19:             if nd.visits = causalPostset(nd) then
20:                 for all ndout ∈ causalPostset(nd) do
21:                     if ¬ndout.Estart.specified ∧ ndout.Efinish.specified then
22:                         ndout.Estart := nd.Efinish
23:                         ndout.Eduration := ndout.Eduration ∩ ndout.Efinish−
24:                         ndout.Lstart
25:                     if ¬ndout.Lstart.specified ∧ ndout.Lfinish.specified then
26:                         ndout.Lstart := nd.Lfinish
27:                         ndout.Lduration := ndout.Lduration ∩ ndout.Lfinish−
28:                         ndout.Estart                                    ▷ Eq.(6.7)
29:             else
30:                 remove nd from NextWorking
31:             Working := NextWorking
```

$$\begin{aligned} [T_{s,e}, T_{s,l}] &= [T_{f,e} - D_l, T_{f,l} - D_e] \\ [T_{f,e}, T_{f,l}] &= [T_{s,e} + D_e, T_{s,l} + D_l] \\ [D_e, D_l] &= [max(0, T_{f,e} - T_{s,l}), T_{f,l} - T_{s,e}]. \end{aligned} \tag{6.6}$$

When comprehensive time information is available for a node, the precision of the information can be enhanced by employing the following equations:

$$\begin{aligned} [T_{s,e}, T_{s,l}] &= [T_{f,e} - D_l, T_{f,l} - D_e] \cap [T_{s,e}, T_{s,l}] \\ [T_{f,e}, T_{f,l}] &= [T_{s,e} + D_e, T_{s,l} + D_l] \cap [T_{f,e}, T_{f,l}] \\ [D_e, D_l] &= [max(0, T_{f,e} - T_{s,l}), T_{f,l} - T_{s,e}] \cap [D_e, D_l] \end{aligned} \tag{6.7}$$

In Figure 6.5(a), the missing is the start time of e_4, and the end time and duration are specified for the node. Therefore, we use the causal relation to estimate the missing start time, the nearest node with specified time information is e_4 the end time and duration are specified. Therefore, the end time of node e_4 is $I_e^{e_4} = [3{:}9]$ minute and second respectively and the duration is $I_d^{e_4} = [d99]$ the computation of start time for e_4 is as follows:

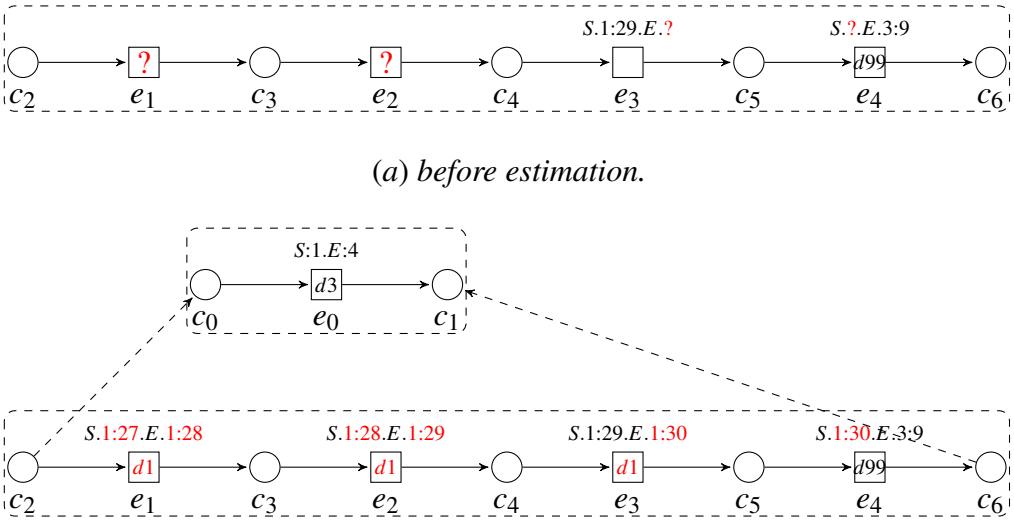
$$[3{:}9 - 99] = [1{:}30]$$

In Figure 6.5, the computation of the missing time information of e_1 and e_2 is carried out using the identified nodes, and performing the *forward DFS* (DFS) procedure to calculate the unspecified time and duration of the nodes causally related to e_3.

The start time of the node e_3 is $I_s^{e_3} = [1{:}29]$ minute and second respectively, and the *defualt duration* is used for nodes with unspecified duration, $I_d^{e_3} = [d1]$, which is added from the specified start time of e_3 using Eq. (6.6), as shown in Figure 6.5(b).

6.7 Consistency checking for TGBSA-nets

The different abstraction levels of the models we consider have different time granularity. Consistency in the contexts of time granularity refers to the uniformity and coherence of the time information or increments used throughout your system or data. It means that the time



(*a*) *before estimation.*

(*b*) *after estimation.*

Figure 6.5 Time granularity before and after estimation (minutes and seconds).

resolution or level of detail remains consistent and aligned across different components or sources of time-related information.

Inconsistent time granularity can lead to confusion, errors, or inaccurate analysis when working with time-based data. Therefore, ensuring consistency is important for accurate interpretation and meaningful comparisons of time-related information.

- **Uniformity of Time Intervals.** It is crucial to verify the consistency of the time intervals or increments employed in the dataset. In particular, this implies that temporal granularity, measured in seconds or minutes, is consistently maintained at the same level of abstraction.

- **Verify Alignment.** Ensure that the time aligns accurately across all levels of granularity. If you have data at the second level, ensure that the time unit at the minute level corresponds to the time unit at the second level.

- **Validate Relationships.** Verify the interconnections among various levels of granularity. An illustration of this would be to verify whether the start and finish times of a higher-level interval encapsulate the corresponding lower-level intervals without any overlapping or gaps.

Let $\updownarrow$ the projection relation of the temporal structure $F_{min\text{-}sec}$ used in the definition of *tgbsan*. The projection relation can be seen as a way of connection or alignment between different levels of time granularity. In the time granularity model of *Seconds* and *Minutes*, the projection relation $\updownarrow$ signifies that there is a consistency between the occurrences of transitions at the upper-level and the corresponding ones at the lower-level.

The consistency of time granularity can be assessed by ensuring the following:

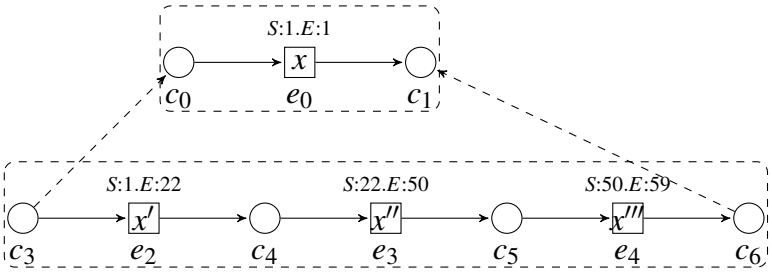
- Let $causal \subseteq T_{lanet} \times T_{hanet}$ be a binary relation consisting of *causally* related pairs of transitions of the TGBSA-net appearing in different levels. The time information of *causal* at different level *minute* and *seconds* is consistent if the following is satisfied:

$$\forall (g,h) \in causal \;\; \exists t \in Seconds\text{: } (T^g_{f,l} \prec t) \wedge \updownarrow (t, T^h_{s,e}) \tag{6.8}$$

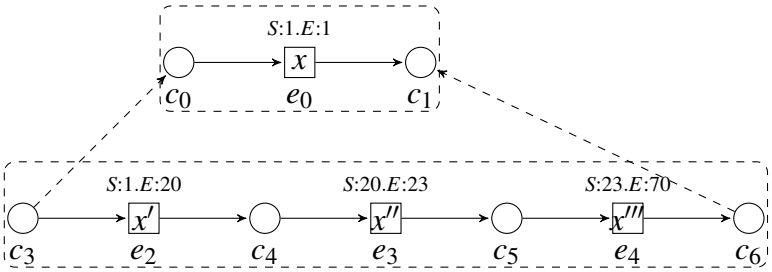
  Intuitively, $T^g_{f,l} \leq T^h_{s,e}$ after a suitable conversion of $T^h_{s,e}$ from minutes to seconds. In other words, Eq.(6.8) states that there two causally related transitions, g in the lower-level and h in the upper-level, should have consistent time information (expressed in seconds for transition g in the lower-level and in minutes for transition h).

Algorithm 14 TGBSA-net consistency

function: Boolean granularity consistency (Relation *causal*)
Result $:= \varnothing$ ▷ inconsistent nodes
for $(t_1, t_2) \in causal$ **do**
  if $t_1.start.early > t_2.start.late \vee t_1.finish.early > t_2.finish.late$ **then**
    add t_1, t_2 to *Result* ▷ Eq.(6.8)
for all transitions t **do**
  if $t = \varnothing \wedge t.start \not\subset\updownarrow (v).start$ **then** ▷ Eq.(6.9)
    add t to *Result*
  else if $t = \varnothing \wedge t.finish \not\subset\updownarrow (v).finish$ **then**
    add t to *Result*
for $t \in$ **T do**
  if ${}^\bullet t.finish \succ t^\bullet.start$ **then** ▷ Eq.(6.10)
    add t to *Result*
return *Result*



(*a*) *consistent time granularity model*



(*b*) *inconsistent time granularity model*

Figure 6.6 Time granularity models

- For all transitions t, v with $(t, v) \in \beta$ and t belonging to the lower-level of the TGBSA-net, the following must hold:

$$I^t \subset I^v \,, \tag{6.9}$$

where *subset* the inclusion relation of the temporal structure $F_{min\text{-}sec}$ used in the definition of *tgbsan*.

- Let t be a transition of the TGBSA-net. After that the following verifies that the finish time of ${}^\bullet t$ is not greater than the start time of $t^\bullet$

$$\forall t{:}(I^{\bullet t}_{f,l} \prec I^{t^\bullet}_{s,e}) \tag{6.10}$$

In Algorithm 14 presented above, line 4 verifies the consistency of all binary relations in *causal*, lines 7 and 9 verify Eq.(6.9) of the TGBSA-net. The return value of the function is all the nodes which are time granularity behaviourally inconsistent.

Transition e_0 in Figure 6.6(a) is node consistent with the transitions on the lower granularity. However, transition e_0 in Figure 6.6(b) is node inconsistent as the second-level time does not match the minute-level time.

Time granularity enables the specification of the temporal behaviour and the necessary properties of the entire system and its components in relation to various time scales. In the literature, various methods for representing and reasoning about time granularity have been proposed. We provide a comprehensive explanation of the main suggestions for the logical framework.

6.8 Tool Support

In the previous sections we introduced theoretical underpinnings and simulation algorithm of the time granularity model for the two-level TGBSA-net. The time annotations in the upper-level acyclic net represent minutes. The lower-level acyclic net shows the detailed behaviour of the system using seconds as time units. Subsequently, this section provides a brief overview of the prototype tool support using Java that has been developed based on the details that are presented in this chapter. We have developed a tool support prototype that includes two levels (for time granularity of minutes and seconds), as well as time simulation capabilities. In addition, adding a time simulation feature lets a user practice detailed temporal behaviours and processes on the two different levels of minutes and seconds, which should make the predictive modelling and scenario analysis easier. As shown in Figure 6.7, the tool enables the entry and display of the start and end time units for each event in the lower-level seconds sequence, along with the display on the middle of each transition of the calculated duration. As illustrated in Figure 6.7, users can both generate and simulate the lower-level *seconds*. In addition, as shown in Figure 6.8, by pressing the "Minutes" button users can effortlessly switch to create the upper level *minutes* with the arrows between the two levels which used to capture the relationships between lower-level and upper-level behaviours, and also a calculated duration in the middle of each transition. Transition e_0 in the upper level abstracts three consecutive transitions in the lower level: e_0, e_1 and e_2. This user-friendly

feature provides users the freedom to investigate temporal dynamics and makes it simple for them to switch between granularities according to their analytical requirements.

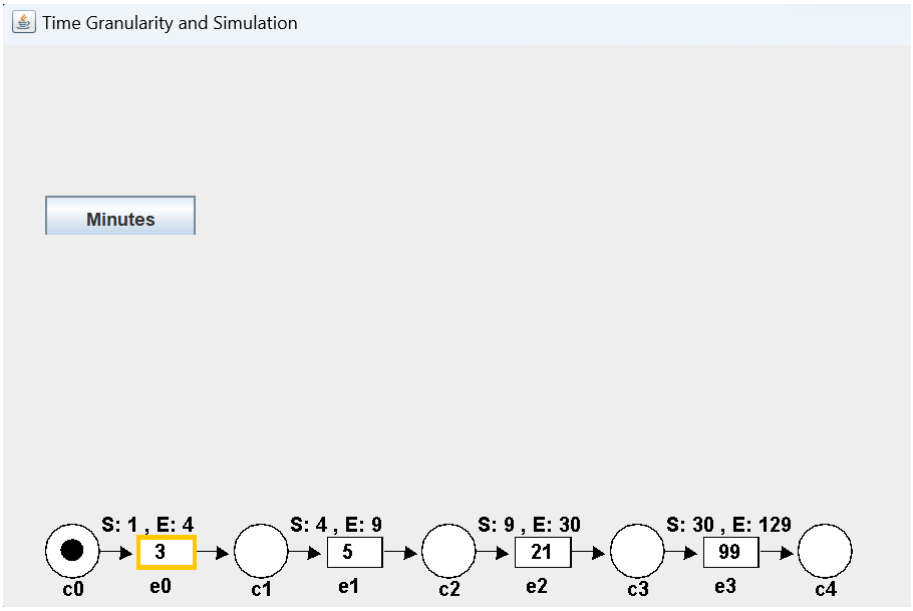


Figure 6.7 Second level.

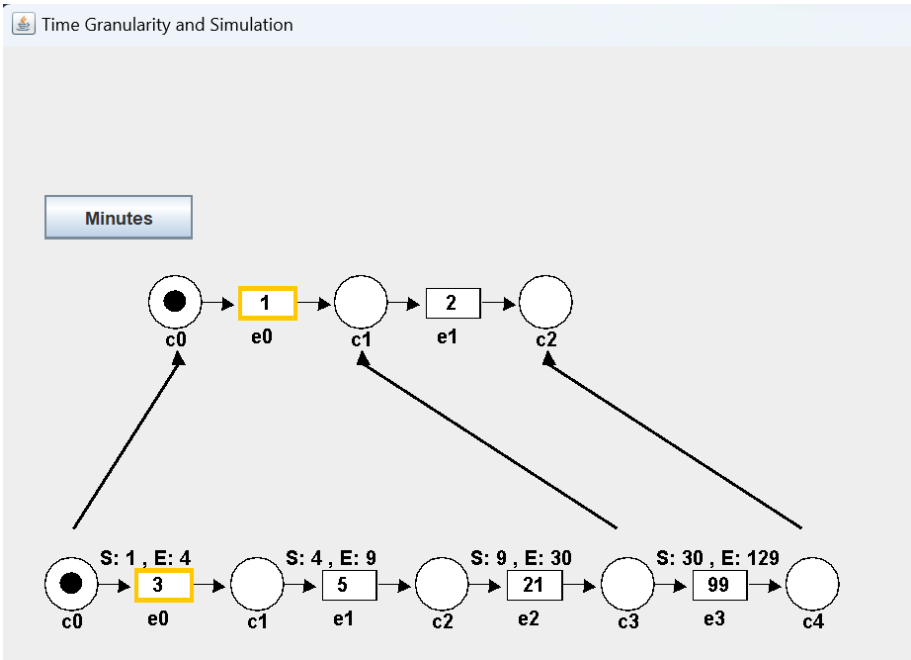


Figure 6.8 Minute level.

6.8.1 Time Granularity Simulation Tool

We developed a time simulation functionality for TGBSA-nets. As shown in Figure 6.9, if the start, end and duration in the lower-level second sequence are specified, then the initial marking is automatically set for the two levels, and all enabled transitions are highlighted. As shown in Figure 6.9, the first enabled transition e_0, has the duration of 1_m and functions at the minute level, which is the higher granularity level. In contrast, the other enabled transition, functioning at the second level (which is the lower granularity level), is also represented as e_0 and has a duration of 1_s. As shown in Figure 6.10, the simulation of TGBSA-nets steps is then conducted automatically by counting down the calculated duration for each transition. The durations for the two levels are represented by different colours: purple for minutes (upper level) and cyan for seconds (lower level). If the token reaches the final event at the lower-level of granularity within an abstracted minute, the system will automatically transfer the token to the output place at the minute level. For example, in Figure 6.10, when e_2 in the lower level second is fired, e_0 in the upper level minute will execute automatically after it. This automated approach provides smooth coordination among the various levels of detail.

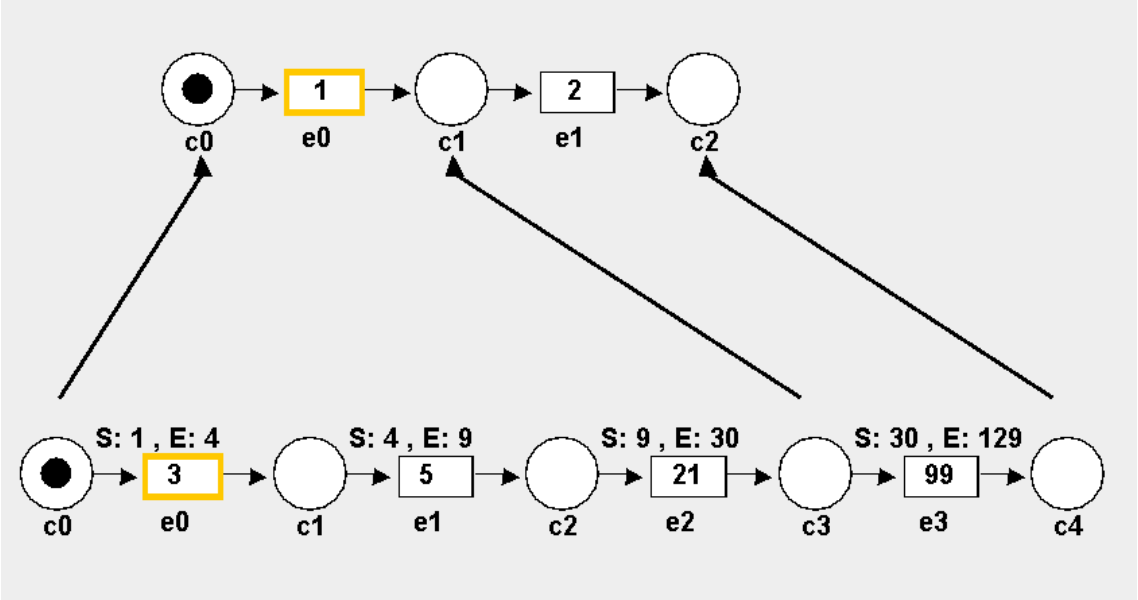


Figure 6.9 Enabled event for simulation.

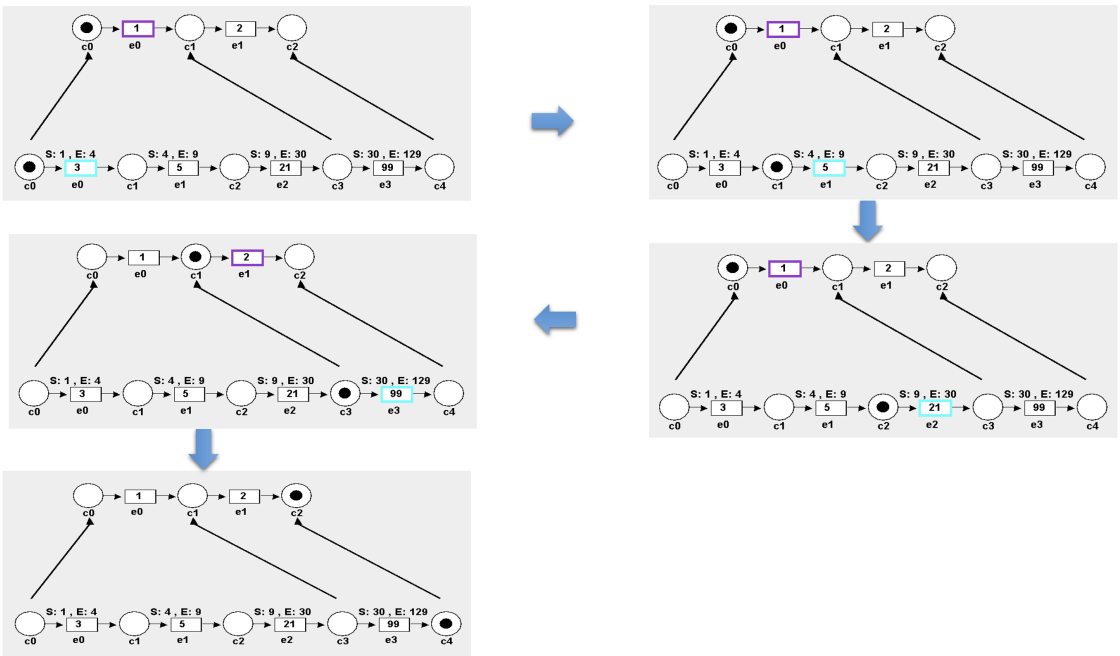


Figure 6.10 Time granularity simulation tool for a TGBSA-net .

6.8.2 Time granularity Visualisation

The time granularity tool enables the display of time unit information for each transition of a TGBSA-net model. Figure 6.11 shows the two levels of the time unit representation of an transition node: the upper-level shows minutes and the lower level shows seconds displayed with start, end and duration. Thus, the time unit information displayed for each transition indicates the start and end times for that transition. In addition, each transition indicates its duration in the middle of the transition node, which the tool calculates from the start and end time unit intervals.

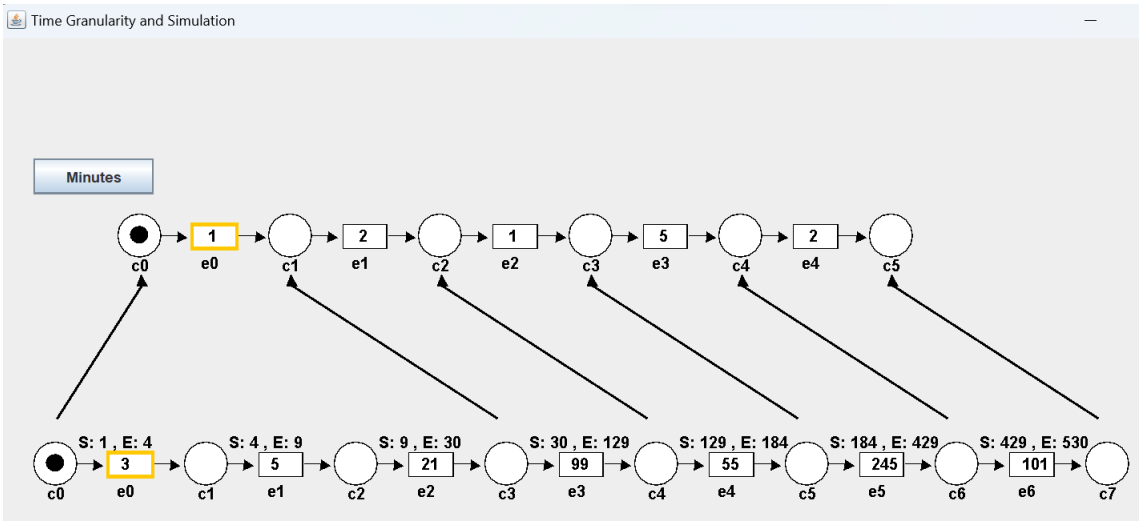


Figure 6.11 TGBSA-net visualisation.

6.8.3 Case Study

In this section, we analyse a scenario of time granularity in two-level TGBSA-net using the tool we have discussed above. The evaluation of the proposed solution and validation of its effectiveness is carried out using a small scenario of a burglar entering a gold-selling shop. The description of the scenario is given below.

"The burglar enters the gold-selling shop at 1:00 PM and for 30 seconds pretends to browse around like a typical customer. He then acts as though he's intrigued by the jewellery displays for a further thirty seconds. He arrives with a gun at 1:01 PM and controls of the shop, which he holds for five minutes. He works rapidly, and complete the task of turning off the shop's security system by 1:06 PM in under 5 minutes. He gives the employees five minutes to gather as much gold and jewellery as they can before telling them to put it in his bag at 1:11 PM. Then, at 1:16 PM, he takes three minutes to ensure that everyone is safe before locking the employees and any customers in a storage area to prevent them from interfering. Then, around 1:19 PM, he searches for a rear door for thirty seconds without getting noticed. After finding the door, he takes an additional thirty seconds to silence any associated alarms. He enters the back door softly at 1:20 PM, and spends the next thirty seconds checking everything to make sure it's secure before going back inside."

The table in Figure 6.12 shows the list of transitions in the scenario with (start, end and duration) in real time information for each transition. Figure 6.13 shows the model of the above scenario using the TGBSA-net tool the upper level shows the *minutes* and lower level shows *seconds*. The sequence of fired steps unfolds as follows:

$$\{e_0, e_1\}\ \{h_0\}\ \{e_2, h_1\}\ \{e_3, h_2\}\ \{e_4, h_3\}\ \{e_5, h_4\}\ \{e_6, e_7\}\ \{h_5\}\ \{e_8, h_6\}.$$

Action	Start Time	End Time	Duration
Enter the gold shop	1:00:00 PM	1:00:30 PM	30 sec
Browse jewelry displays	1:00:30 PM	1:01:00 PM	30 sec
Brandish a gun and take control	1:01:00 PM	1:06:00 PM	5 min
Disable the security system	1:06:00 PM	1:11:00 PM	5 min
Force staff to hand over valuables	1:11:00 PM	1:16:00 PM	5 min
Lock staff and customers in storage room	1:16:00 PM	1:19:00 PM	3 min
Search for back door exit	1:19:00 PM	1:19:30 PM	30 sec
Disable back door alarm	1:19:30 PM	1:20:00 PM	30 sec
Open the back door quietly and Exit through it	1:20:00 PM	1:20:30 PM	30 sec

Figure 6.12 The time of the scenario.

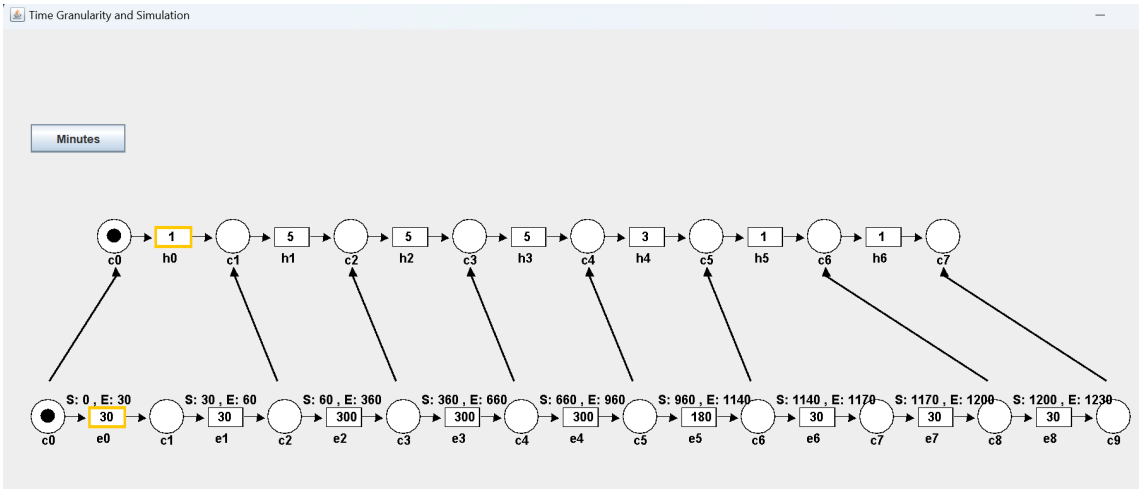


Figure 6.13 The scenario in TGBSA-net with *minutes* and *seconds*.

6.9 Related Work

Time granularity enables the specification of the temporal behaviour and the necessary properties of the entire system and its components in relation to various time scales. Several approaches have been suggested in the literature for representing and analysing time granularity. We describe in some detail the main proposals for the logical framework.

6.9.1 Logical Framework

Temporal logic has been extensively used to deal with temporal knowledge in various domains of computer science [60]. Time granularity has been the subject of extensive research among the numerous papers published on this subject.

The time granularity systems proposed in [30, 56] are quite restrictive. In [30], the author introduces a time granularity system and shows how it can be used to accelerate processes on large temporal databases. The paper outlines a collection of techniques that have been incorporated into a temporal database management system to attain satisfactory performance in applications containing a substantial quantity of known or predictable events and propositions that may endure for periods of time.

The time granularity logics has been implemented in the design of real-time monitoring systems [29], mobile systems [34] and therapy plans in clinical medicine [28].

In [29], the paper focuses specifically on the concept of time granularity, which is one of the most original achieved outcomes. The authors provide a method for describing and managing various levels of temporal precision within a logical specification framework. The research introduces a formal definition of temporal granularity in a logical language used to describe real-time systems. In [28], a logical framework is presented for representing and deducing information regarding various time granularity. A time granularity is defined as a discrete linear time structure in which the beginning and end points of the corresponding granules are appropriately denoted with proposition symbols.

Fuzzy cognitive maps (FCMs) are computational models used in artificial intelligence and cognitive science to represent and simulate complex systems. In [57], an extension to FCMs

is introduced by leveraging a formal language theory called *timed automata*. The theory of *timed automata* allows FCMs to handle a double-layered time granularity more efficiently. This extends the concept of B-time; "B-time is used to diversify the behavior of a given FCM by changing the duration (in terms of seconds, minutes, days, etc.) of a single inference iteration". It shows the iterative nature of a cognitive inference engine is exemplified by this approach, which incorporates model checking tools to evaluate the cognitive and dynamic behaviour of the built framework.

In [39], real-world systems are intricate entities that can be represented at different varying levels of detail. In fact, the concept of a system can be defined recursively, since it is a composition of interacting component systems. The paper demonstrates how a complex system may be characterised at several levels of granularity and how descriptions based on time and events can coexist.

Time granularity has already been the subject of extensive research. This research has been developed using a variety of methods (e.g., set theory, temporal logic, and algebra of relations) and in a number of fields (e.g., artificial intelligence (AI), databases and formal specifications). To the best of our knowledge, all previous related works available in the literature applied time granularity to databases and AI, and have not considered systems such as crime investigation systems. Therefore, in this thesis, we developed a time granularity framework for such systems using Petri nets.

6.10 Conclusion

Time granularity (i.e., the degree of accuracy or precision with which we measure time) is crucial when dealing with systems at various levels of abstraction.

In this chapter, we considered time granularity for BSA-nets, using *Minutes* and *Seconds*. We developed theoretical underpinnings, new algorithms, and implemented a prototype software tool for hierarchical and abstraction-based analysis and simulations of timed behaviours when different time granularity is used at different levels of abstraction. The discussion was carried out using BSA-nets, which are part of the SO-nets framework. The estimation of

missing time was examined in this chapter using the lower level (second) and assessing the consistency between the two levels (minute and second). This chapter's challenge was the introduction of a time granularity model for the BSA-net and the management of multiple levels of time granularity. The idea is to use a suitable temporal structure from Section 6.3 so that time is interpreted in *lanet* and *hanet* using different time scales. A flexible and scalable approach that could accurately represent time across various resolutions was necessary to manage varying levels of granularity.

The implementation of time granularity model for the two-level BSA-nets has also been discussed, we implemented a prototype tool that provides an intuitive exploration of temporal patterns through its user-friendly interface for data visualisation and analysis. The application permitted users to define the time granularity in two distinct levels using BSA-nets. The upper level represented the minute and the lower level represented the second, where the second level corresponded to the minute level. Furthermore, we implemented functionalities that allowed for the simulation of the time granularity behaviour of TGBSA-nets. This facilitated a thorough examination of temporal dynamics. Finally, the prototype tool we have implemented provides a better knowledge of complex systems and how they behave over time.

Chapter 7

Concluding Remarks

7.1 Summary

In this thesis, we developed theoretical underpinnings, algorithms, and tool support for time and time simulation based on SO-nets and their abstractions. These abstractions were derived from the underlying framework of SO-nets, which were introduced in [49].

The development of time properties and time simulation have become essential in contemporary study, providing a powerful method for understanding complex systems and predicting their behaviour under different conditions. By simulating the flow of time, researchers and practitioners can gain insights into how systems might behave in the future and develop strategies for optimising their performance.

A complex evolving system consists of several concurrently acting (sub)systems that interact with the environment and each other, and it can be altered by other systems. Typical examples are large distributed systems whose software is continually updated and dynamically evolving criminal investigations. In this thesis, we developed theoretical underpinning and tool support for time and time simulation based on BSO-nets, CSO-nets and ALTO-net. These abstractions were derived from the underlying framework of SO-nets. In addition, we analysed timed behaviours of CE-systems in the case of different time granularities being used at various levels of abstraction. Furthermore, we extended the SO-nets tool by providing new algorithms and implementations that allow the simulation of timed behaviours of a CE-

systems in meaningful ways. The work delivered an extension of the WORKCRAFT platform, which is a tool that provides a flexible common underpinning for graph based models, and its plugin SONCRAFT which provides some initial facilities for entering, editing, validating and simulating SO-nets. We also extended the concept of the behavioural abstraction of BSA-nets to support various time units at multiple levels.

In Chapter 2, we presented the background about SO-net and their abstractions (behavioural structured occurrence nets, communication structured occurrence nets and alternative occurrence net). Furthermore, we have considered the background of time simulation along with its techniques; there are many time simulation techniques, and we briefly explained them. In addition, a background on time granularity was provided. We discussed the property layers of the hierarchical structure of time granularity, and we analysed the hierarchical structure of temporal granularity, emphasising its different levels and characteristics. We also discussed these layers at various scales.

In Chapter 3, the concept of time property information was added to the basic models based on SO-nets and the ALTO-net model to represent and analyse causally connected events and concurrent occurrences in developing systems. We provided fundamental ideas and notations to display time limits and duration interval for each node in the model. By utilising intervals, this temporal information can be ambiguous or lacking temporal information. Therefore, we discussed algorithms and how time data from a SO-net can be used to estimate and increase the precision of (date-time) intervals using default duration intervals. We also discussed algorithms to check consistency for the basic SO-nets and ALTO-net.

In Chapter 4, we presented timed simulation behaviour for the basic and alternative SO-nets. We also provided a new novel execution time semantics for variants of SO-nets in order to simulate time step-by-step. Timed simulation behaviour is a notion that has been proposed in this chapter to examine systems, such as criminal investigations, to aid investigators in reconstructing a sequence of events and analysing the behaviour of the system. The firing sequences, notations and algorithms of timed-interval simulation SO-nets were presented. The firing of an enabled transition in timed simulation involves two distinct stages: the first stage calculates the duration of the firing time, while the second stage begins the

countdown of the *time units* within the enabled transition's duration interval. We provided a new algorithm and definition for determining maximal firing steps with time semantics in SO-nets. This development is crucial for comprehending the evolution of a system over time, especially in scenarios where numerous processes are concurrently operating.

In Chapter 5, we presented the time property and timed simulation tools that could be used, for example, in criminal investigations to help investigators reconstruct a timeline of events and analyse the behaviour of the systems. The prototype tools were implemented as SONCRAFT plugins, we introduced the ideas and algorithms developed in the formal methods part of Chapters 3 and 4. First, we implemented the time property intervals for each node in the SO-nets and ALTO-net. In addition, we developed time simulation tools to simulate the time behaviours of the model. Further, algorithms checked the consistency and correctness of the new abstraction methods developed in the prototype tools. The tool provided efficient date estimation for events with unspecified date information (estimator setting) and a tool to check the consistency of the (date-time) information for the entire SO-net.

In Chapter 6, we investigated time granularity and developed theoretical underpinnings and new algorithms for hierarchical and abstraction-based analyses and simulations of timed behaviours for cases in which different time granularity was used at different levels of abstraction. We used BSA-net that are part of the SO-nets framework. In this chapter, we discussed algorithms and how time data from a TGBSA-net can be used to estimate and increase the precision of time unit and using default duration. We also discussed algorithms to check consistency for the two levels minutes and seconds of TGBSA-net.

Furthermore, we implemented a prototype software tool for two-level BSA-net and added two time granularities: *minutes* for the upper level and *seconds* for the lower level. Simulations of timed behaviours when different time granularities is used at different levels of abstraction using the algorithms developed in the theoretical investigation were described.

7.1.1 Research objectives

[Developing the time property information for SO-nets.] We extended the existing basic SO-net model and ALTO-net to include time information. We presented a formal description of how to verify the consistency of timing information provided in a SO-net and how to determined the time information estimation of a specific node or the entire SO-net by utilising causal relations.

Extending the concept of time property information. We introduced a time simulation to simulate the time behaviour of the model using SO-nets. New execution time semantics were introduced for variants of SO-net in order to achieve a step-by-step time simulation. We also provided a new algorithm for maximal firing steps with time semantics for SO-nets. We then evaluated the proposed solution and validated its effectiveness by using a small case study constructed by an MSc student [38].

Designing and implementing new tool in SONCRAFT plug-ins. We developed SONCRAFT plug-ins implementing the ideas and algorithms developed in the formal parts of Chapters 3 and 4. The prototype tool implementation and evaluation combines and implements the time property for SO-nets and ALTO-net using specific time simulation algorithms developed in theoretical investigation with suitably adapted simulation algorithms found in other software tools. Furthermore, algorithms checking the consistency and correctness of the new abstraction methods were incorporated into the prototype tool.

Extending the concept of behavioural abstraction to support time granularity. We investigated the time granularity and developed theoretical underpinnings and new algorithms for hierarchical and abstraction-based analyses and simulations of timed behaviours for cases in which different time granularity is used at different levels of abstraction (using BSA-nets that are part of the SO-nets framework). We provided the implementation of the prototype software tool for BSA-nets by adding different time granularities (*minute* and *second*) and simulations of timed behaviours using the algorithms developed in the theoretical investigation.

7.2 Challenges

Several intriguing challenges that emerged throughout the study described in this thesis are as follows:

1. **Developing (date-time) information**

   In Chapter 3, the challenge consisted of adapting algorithms and notations from the work [23] for the purpose of handling dates. Additionally, it was necessary to guarantee that the model appropriately reflects the nuances of time-based information, which is essential for applications such as criminal investigations.

2. **SONCraft: Tool for SO-nets with Time**

   Initially, comprehending the current codebase is essential to recognise possible integration points and dependencies. Analysing the tool's structure, functionality and design patterns thoroughly to ensure compatibility with the new code. Incorporating a time property and time simulation tools into existing code involves more than just linking new components with existing ones. It involves not only getting it to function but also ensuring it keeps accurate time. This difficulty encompasses both technical aspects, such as ensuring the new tool does not disrupt the existing code, and the task of synchronising the simulation time with time information.

3. **Dealing with Time Granularity in BSA-nets**

   Incorporating multi-level time granularity into BSA-nets poses significant challenges. The intricate representation of several levels of abstraction in a BSA-nets structure might result in heightened model complexity, which can make it more difficult to understand and analyse. Furthermore, ensuring the consistency and coherence of multi-level BSA-nets, especially when handling interactions between different levels of abstraction, is challenging in terms of maintaining model integrity and accurately representing system behaviour.

7.3 Future Work

The following are some suggestions for future research:

1. **Adding Spatial Information to the Nodes**

   Integrating spatial information into nodes that already include time information improves the contextual understanding of temporal data. This integration enhances the comprehension and analysis of the progression of events across both the temporal and spatial dimensions.

2. **Extending the Time Simulation by Adding a Filter Simulation**

   Filtering is of the utmost importance in simulation, particularly when dealing with complicated models because it helps to streamline the data and concentrate on the features of the system being replicated that are the most significant. For future work, we can consider the addition of a filtering tool for time to simulate the behaviour of the model within a range of time. We can improve the usefulness and dependability of the simulation findings while gaining a deeper understanding of complicated systems by utilising the proper filtering strategies. We can add a function that allows the specification of the start and end (date-time) information in the simulation model prior to its start.

3. **Adding Multiple Levels to BSA-nets**

   The future work will add multi-level modelling of *year*, *month*, *day* and *hour*. A key problem of handling complexity when large datasets are involved will be addressed by developing a hierarchical approach to simulation based on behavioural abstractions. In Chapter 6 regarding time granularity, we introduced two levels of time units: *minutes* and *seconds*. For the work that will be done in the future, we might consider adding a multi-level approach to time granularity that would include *year*, *month*, *day* and *hour*. An examination of temporal data at different levels of detail would be required for this. We can acquire a more comprehensive grasp of the temporal dynamics and better capture the nuances that are present in the data if we incorporate numerous layers of

time granularity into our analysis. This strategy has the potential to deepen the scope of the analysis and offer insightful information regarding the temporal features of the research topic that we are investigating.

Adding multiple levels to BSA-nets would significantly enhance the practical applicability of the framework developed in this thesis. In particular, the multi-level aspects could greatly improve the range of the models handled by the tool. First, the formal analyses would be applicable to much larger cases, making the implemented tool applicable to particularly suitable for areas such as software engineering project management (e.g., using the PERT diagrams) or asynchronous circuit design (a primary application area of the Workcraft toolset) Secondly, the visualisation and simulation of the models investigated would be allow the resulting tool to communicate and demonstrate to the users the behavioural information at progressively more abstract (succinct) levels. This could, in particular, help in conducting criminal/accident investigations, as different teams involved could access the information explained in terms of events matching in the best possible way their expertise/interest.

4. **Time Granularity Tool Support**

   In future work, the objective is to integrate the tool support created in Chapter 6 for BSA-nets into the SONCRAFT architecture, while also expanding its functionalities. Our objective is to evolve the tool such that it can handle both synchronous and asynchronous communication structured acyclic nets over numerous hierarchical layers. By incorporating these characteristics, we expect to facilitate a more thorough examination of intricate systems, thereby helping investigators to efficiently simulate and analyse complex network structures. This integration is an important milestone in developing a more powerful and flexible collection of tools for analysing and designing systems in a number of contexts, such as crime investigation.

   Finally, it would be interesting to develop a generic tool-supported framework where the user could instantiate their particular time model (including time granularity), and

the tool would still be able to support consistency checking and simulation of the models.

Bibliography

[1] Acampora, G. and Loia, V. (2011). On the temporal granularity in fuzzy cognitive maps. *IEEE Transactions on Fuzzy Systems*, 19(6):1040–1057.

[2] Ajmone Marsan, M., Conte, G., and Balbo, G. (1984). A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS)*, 2(2):93–122.

[3] Alahmadi, M. (2021). Master channel places for communication structured acyclic nets. In *International Workshop on Petri Nets and Software Engineering (PNSE)*, volume 2907 of *CEUR Workshop Proceedings*, pages 233–240. CEUR-WS.org.

[4] Alahmadi, M., Alharbi, S., Alharbi, T., Almutairi, N., Alshammari, T., Bhattacharyya, A., Koutny, M., Li, B., and Randell, B. (2024). Structured acyclic nets. *CoRR*, abs/2401.07308.

[5] Alharbi, S. (2023). Hierarchical simulation of timed behaviours of structured occurrence nets. In *International Workshop on Petri Nets and Software Engineering (PNSE)*, volume 3430 of *CEUR Workshop Proceedings*, pages 143–166. CEUR-WS.org.

[6] Alharbi, S. (2024). Time granularity in behavioural structured acyclic nets. In *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 1–12. IEEE.

[7] Alharbi, T. (2016). *Analysing and visualizing big data sets of crime investigations using Structured Occurrence Nets*. PhD thesis, School of Computing Science, Newcastle University.

[8] Alharbi, T. and Koutny, M. (2018). Visualising data sets in structured occurrence nets. In *PNSE@ Petri Nets/ACSD*, pages 121–132.

[9] Allen, J. F. (1991). Time and time again: The many ways to represent time. *International Journal of Intelligent Systems*, 6(4):341–355.

[10] Almutairi, N. (2022). Probabilistic communication structured acyclic nets. In *International Workshop on Petri Nets and Software Engineering (PNSE)*, volume 3170 of *CEUR Workshop Proceedings*, pages 168–187. CEUR-WS.org.

[11] Almutairi, N. and Koutny, M. (2021). Verification of communication structured acyclic nets using SAT. In *International Workshop on Petri Nets and Software Engineering (PNSE)*, volume 2907 of *CEUR Workshop Proceedings*, pages 175–194. CEUR-WS.org.

[12] Alshammari, T. (2022). Towards automatic extraction of events for SON modelling. In *International Workshop on Petri Nets and Software Engineering (PNSE)*, volume 3170 of *CEUR Workshop Proceedings*, pages 188–201. CEUR-WS.org.

[13] Babulak, E. and Wang, M. (2010). Discrete event simulation. *Aitor Goti (Hg.): Discrete Event Simulations. Rijeka, Kroatien: Sciyo*, page 1.

[14] Badaloni, S. and Berati, M. (1994). Dealing with time granularity in a temporal planning system. In *International Conference on Temporal Logic*, pages 101–116. Springer.

[15] Banks, J. (1998). *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons.

[16] Banks, J. (1999). Introduction to simulation. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*, pages 7–13.

[17] Becher, G., Clérin-Debart, F., and Enjalbert, P. (2000). A qualitative model for time granularity. *Computational intelligence*, 16(2):137–168.

[18] Bélanger, J., Venne, P., Paquin, J.-N., et al. (2010). The what, where and why of real-time simulation. *Planet Rt*, 1(1):25–29.

[19] Berthomieu, B. and Menasche, M. (1983). An enumerative approach for analyzing time petri nets. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congress*, pages 41–46. North-Holland/IFIP.

[20] Best, E. and Devillers, R. (1987). Sequential and concurrent behaviour in petri net theory. *Theoretical Computer Science*, 55(1):87–136.

[21] Best, E. and Fernández, C. (1988). *Nonsequential Processes - A Petri Net View*, volume 13 of *EATCS Monographs on Theoretical Computer Science*. Springer.

[22] Best, E. and Fernández, C. (2012). *Nonsequential processes: a Petri net view*, volume 13. Springer Science & Business Media.

[23] Bhattacharyya, A., Li, B., and Randell, B. (2016). Time in structured occurrence nets. In *International Workshop on Petri Nets and Software Engineering (PNSE)*, volume 1591 of *CEUR Workshop Proceedings*, pages 35–55. CEUR-WS.org.

[24] Bruzzone, A. G. and Massei, M. (2017). Simulation-based military training. *Guide to Simulation-Based Disciplines: Advancing Our Computational Future*, pages 315–361.

[25] Cassandras, C. G. and Lafortune, S. (2008). *Introduction to discrete event systems*. Springer.

[26] Cerone, A. and Maggiolo-Schettini, A. (1999). Time-based expressivity of time Petri nets for system specification. *Theoretical Computer Science*, 216(1-2):1–53.

[27] Chiola, G. and Ferscha, A. (1993). Distributed simulation of Petri nets. *IEEE Parallel and Distributed Technology*, 1(3):33–50.

[28] Combi, C., Franceschet, M., and Peron, A. (2002). A logical approach to represent and reason about calendars. In *Proceedings Ninth International Symposium on Temporal Representation and Reasoning*, pages 134–140. IEEE.

[29] Corsetti, E., Montanari, A., and Ratto, E. (1991). Dealing with different time granularities in formal specifications of real-time systems. *Real-Time Systems*, 3(2):191–215.

[30] Dean, T. (1989). Using temporal hierarchies to efficiently maintain large temporal databases. *Journal of the ACM (JACM)*, 36(4):687–718.

[31] Emerson, E. A. (1990). Temporal and modal logic. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 995–1072. Elsevier and MIT Press.

[32] Euzenat, J. and Montanari, A. (2005). Time granularity. In Fisher, M., Gabbay, D. M., and Vila, L., editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, volume 1 of *Foundations of Artificial Intelligence*, pages 59–118. Elsevier.

[33] Fadeyi Johnson, A. (2013). Modeling and simulation of a time series forecasting models and analysis in a manufacturing industry. *International Journal of Scientific and Engineering Research (IJSER)*, 4(9):516–529.

[34] Franceschet, M., Montanari, A., de Rijke, M., et al. (2000). Model checking for combined logics. In *Proceedings of the 3rd International Conference on Temporal Logic (ICTL)*, pages 65–73.

[35] Fujimoto, R. M. (2000). *Parallel and distributed simulation systems*, volume 300. Citeseer.

[36] Gardey, G., Lime, D., Magnin, M., and Roux, O. H. . (2005). Romeo: A tool for analyzing time Petri nets. In *Computer Aided Verification: 17th International Conference*, pages 418–423. Springer.

[37] Guo, Z., Zhang, Y., Zhao, X., and Song, X. (2017). A timed colored Petri net simulation-based self-adaptive collaboration method for production-logistics systems. *Applied Sciences*, 7(3):235.

[38] Hand, F. (2019). An evaluation of structured occurrence nets for crime investigation. Master's thesis, School of Computing Newcastle University.

[39] Hobbs, J. R. (1990). Granularity. In *Readings in qualitative reasoning about physical systems*, pages 542–545. Elsevier.

[40] Horváth, Á. and Molnár, A. (2015). Tipeness: A timed Petri net simulator software with generally distributed firing delays. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 3(8).

[41] Iannoni, A. P. and Morabito, R. (2006). A discrete simulation analysis of a logistics supply system. *Transportation Research Part E: Logistics and Transportation Review*, 42(3):191–210.

[42] Ingalls, R. G. (2011). Introduction to simulation. In *Proceedings of the 2011 winter simulation conference (WSC)*, pages 1374–1388. IEEE.

[43] Jensen, K., Kristensen, L. M., Jensen, K., and Kristensen, L. M. (2009). Formal definition of timed coloured Petri nets. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, pages 257–271.

[44] Kawises, J. and Vatanawood, W. (2019). Formalizing time Petri nets with metric temporal logic using promela. In *20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 162–166. IEEE.

[45] Kleijn, J. and Koutny, M. (2011). *Causality in structured occurrence nets*. Springer.

[46] Koutny, M. and Randell, B. (2009a). Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. *Fundam. Inform.*, 97(1-2):41–91.

[47] Koutny, M. and Randell, B. (2009b). Structured occurrence nets: incomplete, contradictory and uncertain failure evidence. Technical report, School of Computing Science, Newcastle University.

[48] Li, B. (2012). Verification and simulation tool for communication structured occurrence nets. *Newcastle University, School of Computing Science Technical Report Series, CS-TR-1363*.

[49] Li, B. (2017). *Visualisation and Analysis of Complex Behaviours using Structured Occurrence Nets*. PhD thesis, School of Computing Science, Newcastle University.

[50] Li, B., Randell, B., Bhattacharyya, A., Alharbi, T., and Koutny, M. (2018). Soncraft: A tool for construction, simulation, and analysis of structured occurrence nets. In *18th International Conference on Application of Concurrency to System Design (ACSD)*, pages 70–74. IEEE.

[51] Maier, J. F., Eckert, C. M., and Clarkson, P. J. (2017). Model granularity in engineering design–concepts and framework. *Design Science*, 3:1.

[52] Merlin, P. and Farber, D. (1976). Recoverability of communication protocols-implications of a theoretical study. *IEEE transactions on Communications*, 24(9):1036–1043.

[53] Missier, P., Randell, B., and Koutny, M. (2012). Modelling provenance using structured occurrence networks. In Groth, P. and Frew, J., editors, *Provenance and Annotation of Data and Processes - 4th International Provenance and Annotation Workshop, IPAW 2012, Santa Barbara, CA, USA, June 19-21, 2012, Revised Selected Papers*, volume 7525 of *Lecture Notes in Computer Science*, pages 183–197. Springer.

[54] Molloy, M. K. (1985). Discrete time stochastic Petri nets. *IEEE Trans. Software Eng.*, 11(4):417–423.

[55] Montanari, A. (1996). *Metric and Layered Temporal Logic for Time Granularity*. PhD thesis, University of Amsterdam.

[56] Montanari, A., Maim, E., Ciapessoni, E., and Ratto, E. (1992). Dealing with time granularity in the event calculus. In *FGCS*, pages 702–712.

[57] Montanari, A. and Policriti, A. (1996). Decidability results for metric and layered temporal logics. *Notre Dame Journal of Formal Logic*, 37(2):260–282.

[58] Mota, E., Haggith, M., Smaill, A., and Robertson, D. (1995). *Time granularity in simulation models of ecological systems*. University of Edinburgh, Department of Artificial Intelligence.

[59] Mota, E., Robertson, D., and Smaill, A. (1996a). Naturetime: temporal granularity in simulation of ecosystems. *Journal of Symbolic Computation*, 22(5-6):665–698.

[60] Mota, E., Robertson, D., and Smaill, A. (1996b). Naturetime: temporal granularity in simulation of ecosystems. *Journal of Symbolic Computation*, 22(5-6):665–698.

[61] Özgün, O. and Barlas, Y. (2009). Discrete vs. continuous simulation: When does it matter. In *Proceedings of the 27th international conference of the system dynamics society*, volume 6, pages 1–22.

[62] Pelz, E. (2016). Timed processes of interval-timed Petri nets. In Schlingloff, B., editor, *Proceedings of the 25th International Workshop on Concurrency, Specification and Programming, Rostock, Germany, September 28-30, 2016*, volume 1698 of *CEUR Workshop Proceedings*, pages 13–24. CEUR-WS.org.

[63] Petri, C. A. and Reisig, W. (2008). Petri net. *Scholarpedia*, 3(4):6477.

[64] Pfahl, D. (2006a). Prosim/ra — software process simulation in support of risk assessment. *Value-Based Software Engineering*, pages 263–286.

[65] Pfahl, D. (2006b). Prosim/ra—software process simulation in support of risk assessment. In *Value-based software engineering*, pages 263–286. Springer.

[66] Poole, D. and Raftery, A. E. (2000). Inference for deterministic simulation models: the bayesian melding approach. *Journal of the American Statistical Association*, 95(452):1244–1255.

[67] Popova-Zeugmann, L. (1991). On time Petri nets. *J. Inf. Process. Cybern.*, 27:227–244.

[68] Ramchandani, C. (1974). *Analysis of asynchronous concurrent systems by Petri nets*. PhD thesis, Massachusetts Institute of Technology.

[69] Randell, B. (2013). Incremental construction of structured occurrence nets. *Newcastle University, School of Computing Science Technical Report Series, CS-TR-1384*.

[70] Randell, B. and Koutny, M. (2007). Failures: Their definition, modelling and analysis. In *Theoretical Aspects of Computing - ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, volume 4711 of *Lecture Notes in Computer Science*, pages 260–274. Springer.

[71] Schreiber, F. A. et al. (1991). State and time granularity in systems description: An example. *Real Time Systems Newsletter*, 7(3):12–17.

[72] Shannon, R. E. (1975). Systems simulation: The art and science prentice hall. *New Jersey*.

[73] Sharma, P. (2015). Discrete-event simulation. *Int. J. Sci. Technol. Res*, 4(04):136–140.

[74] Shreckengost, R. C. (1985). Dynamic simulation models: How valid are they? *NIDA research monograph*, 57:63–70.

[75] Silva, J. R. and Del Foyo, P. M. (2012). Timed Petri nets. In *Petri Nets: Manufacturing and Computer Science*, pages 359–378. InTech.

[76] Swinerd, C. and McNaught, K. R. (2012). Design classes for hybrid simulations involving agent-based and system dynamics models. *Simulation Modelling Practice and Theory*, 25:118–133.

[77] Thong, W. J. and Ameedeen, M. (2015). A survey of Petri net tools. In *Advanced Computer and Communication Engineering Technology: Proceedings of the 1st International Conference on Communication and Computer Engineering*, pages 537–551. Springer.

[78] van der Aalst, W. M. (1993). Interval timed coloured Petri nets and their analysis. In *Application and Theory of Petri Nets 1993: 14th International Conference Chicago, Illinois, USA, June 21–25, 1993 Proceedings 14*, pages 453–472. Springer.

[79] Wang, M., Wang, S., Guo, J., and Jia, W. (2023). Improving stock trend prediction with pretrain multi-granularity denoising contrastive learning. *Knowledge and Information Systems*, pages 1–28.

[80] Wiederhold, G., Jajodia, S., and Litwin, W. (1991). Dealing with granularity of time in temporal databases. In *International Conference on Advanced Information Systems Engineering*, pages 124–140. Springer.

[81] Workcraft (2018). `https://workcraft.org/`.

[82] Zuberek, W. M. (1980). Timed Petri nets and preliminary performance evaluation. In *Proceedings of the 7th annual Symposium on Computer Architecture*, pages 88–96.